C5

Page 4 File Index Fri Jan 04 15:38:13 2008

Page 4 File Index Fri Jan 04 15:38:13 2008

```
 %/*
 %** Copyright 1997,1998 EMC Corporation
 %*/

 /*
 **  Leading % causes rpcgen to pass a line directly thought to the output,
 **  ie restore_engine.h in this case.  This allows the .h to make a little
 **  more sense and be properly documented.
 */

 %/*
 % * restore_engine.x : EDM Restore Engine C/S communication module
 % *
 % * Mission Statement:  This is an RPCGEN file which defines the RPC interface
 % *                     between the Restore Engine server (which resides on
 % *                     the EDM server) and  the backup client callers of its
 % *                     functions.  This defines the RPC level calls that a
 % *                     "caller" can make and the "service" will respond to.
 % *
 % * Primary Data Acted On: This defines the data that will flow over the wire.
 % *                        The RPC mechanism will take care of data
                                                                marshalling
 % *
 % *
 % * Compile-Time Options:
 % *                     This acutally gets run through RPCGEN not compiled.
                                                                        It
 % *                      must be run through with the -h flag to create a
 % *                      header, the -m flag to create the service side
 % *                      routines, the -l flag to create the client side
 % *                      routines, and the -c flag to create the common data
 % *                      marshalling routines.
 % *
 % * Basic idea here:
 % *                     Define the RPC level interfaces to the Restore Engine
 % *                     and all data types that will be passed via RPC.
 %*/

/* for sharing of STRING(x) and OPAQUE(x) */
#define  IN_DOTX
#include <restore/restoreRPC.h>

%#include <restore/dispatch_daemon.h>

/*****************************************************************
 Constant Definitions
*****************************************************************/

/*****************************************************************
 Enum Definitions
*****************************************************************/

/*****************************************************************
 Typedef Definitions
*****************************************************************/

typedef int       RE_errno_ty;

/*****************************************************************
 Data Structure Definitions
*****************************************************************/

/* Structure to start every RPC request and response - for debug purposes */
        struct RE_rpc_objID
        {
                unsigned long   rpc_type;
```



```
                                                /* RPC Object ID (ie, rpc #) */
                RSTRPC_time_ty  time;   /* creation time */
                long            len;
                                        /* Length of structure, version num? */
        };

        struct RE_null_args {
                RE_rpc_objID RPCobjID;
        };

        struct RE_status_result {
                RE_rpc_objID RPCobjID;
                RE_errno_ty status;
        };

        struct RE_boolean_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_bool     boolResult;
        };

        union RE_restorable_obj switch (RSTRPC_ObjectLevel objLevel)
        {
        case RSTRPC_tlo_type:
                RSTRPC_top_level_obj            *tloInfo;
        default:  /* anything else means NOT tlo -- i.e. container or leaf */
                RSTRPC_user_restorable_object   *uroInfo;
        } ;

const MAX_CHOICE_TEXT=80;

struct Choices {
    bool        isset;
    string      ctext<>;
    Choices     *nextchoice;
};

/* Question types */
const QTYPE_BOOL =      1;
const QTYPE_RAD =       2;
const QTYPE_MULTI =     4;
const QTYPE_STR =       8;
const QTYPE_YESNO =     16;
const QTYPE_INT =       32;

struct Question {
    int         qnum;
    int         qtype;
    int         maxlen;
    int         minlen;
    int         numchoices;
    string      invalidchars<>;
    string      headertxt<>;
    string      qtext<>;
    Choices     *choices;
};

struct Answer {
    int         qnum;
    string      ctext<>;
    Answer      *nextanswer;
};

struct AnswerList {
    int         numanswers;
    Answer      *firstanswer;
```

```
);


/* structures for input and output of re_initialize rpc call:  */

        struct RE_initialize_args {
                RE_rpc_objID RPCobjID;
                string username<>;
        };

/* structures for input and output of get_source_hosts and
 * get_destination_hosts rpc calls:
 */
        struct RE_get_hosts_args {
                RE_rpc_objID    RPCobjID;
                string          hostname<>;
                                                /* only for get_source_hosts */
                short           maxEntries;
                long            cookie;
        };

        struct RE_get_hosts_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                short           numEntries;     /* redundant but useful ? */
                long            cookie;
                RSTRPC_name_list        *hosts;  /* link to first hostname */
        };

/* structure for single character string argument */
        struct RE_string_args {
                RE_rpc_objID    RPCobjID;
                string  name<>;
        };

/* structure for GetHostPlatformType results */
        struct RE_get_host_platform_type_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                int             ptype;
        };

/* structures for input and output of submit RPCs
 */
        struct RE_submit_args {
                RE_rpc_objID    RPCobjID;
                string          hostname<>;
                string          directory<>;
                int             overwritePolicy;
                bool            inPlace;
                int             transport;
                int             submitObjectID;
                int             socketPort;
                string          socketClientName<>;
                string                  mapFile_env<>;
        };

        struct RE_get_submit_results_args {
                RE_rpc_objID    RPCobjID;
                bool            interrupt;
        };

        struct RE_get_submit_results_output {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                int             submitObjectID;
```



```
                                        /* handle for submit object */
                u_long          objectsDone;
        };

/* structures for input of Start RPC
 */
        struct RE_start_args {
                RE_rpc_objID    RPCobjID;
                int             submitObjectID;
                                        /* handle for submit object */
        };

/* structures for input and output of get_restore_feedback RPC
 */
        struct RE_get_restore_feedback_args {
                RE_rpc_objID    RPCobjID;
                bool            quit_restore;   /* flag to request cancel */
        };

        struct RE_Notification {
                int             msgtype;
                int             sourcemodule;
                int             level;
                int             msglen;
                string          msgext<>;
                RE_Notification         *next;
        };

        struct RE_get_restore_feedback_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
               EDMStats         rstStats;
               RE_Notification  *notify;
        };

/* structure for output of get_question RPC
 */
        struct RE_get_question_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                Question        *query;
        };

/* structure for input of set_user_answer RPC
 */
        struct RE_set_user_answer_args {
                RE_rpc_objID    RPCobjID;
                AnswerList      answers;
        };

/* structures for input and output of get_top_level_objects RPC
 */
        struct RE_get_top_level_objects_args {
                RE_rpc_objID    RPCobjID;
                string          sourceHost<>;
                short           maxEntries;
                long            cookie;
        };

        struct RE_get_top_level_objects_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_tlo_list *topLevelObjs;  /* linked list */
                short           numEntries;
                long            cookie;
        };
```

```
/* structures for input and output of get_workitem_templates rpc call:
 */
        struct RE_get_top_level_templates_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_top_level_obj *topLevelObj;
                short           maxEntries;
                long            cookie;
        };

        struct RE_get_top_level_templates_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                short           numEntries;     /* redundant but useful ? */
                long            cookie;
                RSTRPC_name_list *templates;    /* link to first template */
        };

/* structure for input of does_alternate_exist rpc call:
 */
        struct RE_does_alternate_exist_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_top_level_obj *topLevelObj;
                string          templateName<>;
        };

/* structures for input and output of get_restorable_objects RPC's:
 */
        struct RE_get_restorable_objects_start_args {
                RE_rpc_objID    RPCobjID;
                RE_restorable_obj *parentObj;
                long            cookie;
                short           maxEntries;
                bool            allowBadFiles;
        };

        struct RE_get_restorable_objects_start_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
        };

        struct RE_get_restorable_objects_output_args {
                RE_rpc_objID    RPCobjID;
                short           maxEntries;
        };

        struct RE_get_restorable_objects_output_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_uro_list *childrenObjs;  /* linked list */
                long            numEntries;
                long            cookie;
        };

/* structures for input and output of find_restorable_objects RPC's:
 */
        struct RE_search_criteria {
                string  startDirectory<256>; /* Dir to start searching */
                bool    descendDirectory; /* Flag to descend into subdirs */
                string  searchString<128>;   /* String to search for */
                bool    excludeString;   /* Flag to include or exclude */
                RSTRPC_enum_ty  typeOfFile;
                                        /* Types of files to search for */
                string  owner<64>;          /* Specific owner of files */
                bool    excludeOwner;    /* Flag to exclude owner */
                string  group<64>;          /* Specific group of files */
```



```
                bool    excludeGroup;    /* Flag to exclude group */
                RSTRPC_u_hyper sizeInBytes;
                                        /* Specific size of files to find */
                RSTRPC_enum_ty  sizeMatch;
                                        /* type of matching to do for size */
                RSTRPC_time_ty  startTime;
                                        /* First backup date to use */
                RSTRPC_time_ty  endTime;        /* Last backup date to use */
        };

        struct RE_find_restorable_objects_args {
                RE_rpc_objID    RPCobjID;
                RE_search_criteria *searchCriteria;
        };

        struct RE_find_restorable_objects_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
        };

        struct RE_get_find_results_args {
                RE_rpc_objID    RPCobjID;
                bool            interrupt;      /* flag to request cancel */
                short           maxEntries;
                long            cookie;
        };

        struct RE_get_find_results_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_found_obj_list   *foundObjs;     /* linked list */
                long            numEntries;
                long            cookie;
        };

/* structures for input and output of mark_object RPC's:
 */
        struct RE_mark_object_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_user_restorable_object *thisObj;
                RSTRPC_time_ty  backupTime;
                bool            allowBadFiles;
                bool            descend;
        };

        struct RE_mark_object_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
        };

        struct RE_get_mark_results_args {
                RE_rpc_objID    RPCobjID;
                bool            interrupt;      /* flag to request cancel */
        };

        struct RE_get_mark_results_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                u_long          badFileCount;
                u_long          permDenyFileCount;
                u_long          dirMarkCount;
                u_long          fileMarkCount;
                u_long          otherMarkCount;
        };

/* structures for input and output of unmark_object RPC's:
```

```
 */
        struct RE_unmark_object_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_user_restorable_object *thisObj;
                RSTRPC_time_ty  backupTime;
                bool            badFilesOnly;
                bool            descend;
        };

        struct RE_get_unmark_results_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                u_long          badFileCount;
                u_long          dirMarkCount;
                u_long          fileMarkCount;
                u_long          otherMarkCount;
        };

/* structure for output of get_marked_total_size RPC:
 */
        struct RE_get_marked_total_size_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_u_hyper  total;
        };

/* structure for output of get_current_template RPC:
 */
        struct RE_get_current_template_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                string          templateName<>;
                RSTRPC_bool     alternate;
        };

/* structure for output of get_current_backup_time RPC:
 */
        struct RE_get_current_backup_time_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_time_ty  backupTime;
        };

/* structure for input and output of get_all_backup_times RPC:
 */
        struct RE_get_all_backup_times_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_time_ty  startTime;
                RSTRPC_time_ty  endTime;
                RSTRPC_backup_flags_ty flags;
                long            maxEntries;
                long            cookie;
        };

        struct RE_get_all_backup_times_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_time_list *backupTimes;
                long            numEntries;
                long            cookie;
        };

/* structure for input of is_there_xxxx_backup_for_time and
                                                        set_backup_for_time
 * RPC's:
 */
```



```
        struct RE_backup_for_time_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_time_ty  time;
                RSTRPC_backup_flags_ty flags;
        };

/* structure for input of set_'relative'backup * RPC's: */
        struct RE_set_backup_time_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_backup_flags_ty flags;
        };

/* structure for input and output of get_necessary_media RPC:
 */
        struct RE_get_necessary_media_args {
                RE_rpc_objID    RPCobjID;
                long            maxEntries;
                RSTRPC_bool     all;
                long            cookie;
        };

        struct RE_get_necessary_media_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_media_list *mediaList;
                short           numEntries;
                long            cookie;
        };

/* structures for input and output of is_object_markable RPC:
 */
        struct RE_is_object_markable_args {
                RE_rpc_objID            RPCobjID;
                RSTRPC_user_restorable_object   *thisObject;
        };

        struct RE_is_object_markable_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                bool            markable;
        };

/* structures for input and output of is_object_marked RPC:
 */
        struct RE_is_object_marked_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_uro_list *objList;
                u_long          numEntries;
        };

        struct RE_is_object_marked_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                u_long          numMarked;
                RSTRPC_bool     marked<>;
        };

/* structures for input and output of is_object_searchable and
 * get_backup_times_support RPCs:
 */
        struct RE_tlo_query_args {
                RE_rpc_objID            RPCobjID;
                RSTRPC_top_level_obj    *topLevelObj;
        };

        struct RE_catalog_info {
```

```
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                string          level<>;
                string          numrec<>;
                string          catType<>;
        );


/* structure for inputs that require only time
 */
        struct RE_time{
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_time_ty  backupTime;
        );


        struct RE_recx_file_info{
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_recx_file_info fileinfo;
        );

program EDM_RESTORE_ENGINE {

   version EDMRE_FUNCTIONS {

           /* rpc for EDMRST_Initialize */
           RE_status_result
               re_initialize( RE_initialize_args ) = 1;

           /* rpc for EDMRST_GetSourceHosts */
           RE_get_hosts_result
               re_get_source_hosts( RE_get_hosts_args ) = 2;

           /* rpc for EDMRST_GetTopLevelObjects */
           RE_get_top_level_objects_result
               re_get_top_level_objects( RE_get_top_level_objects_args ) = 3;

           /* rpc for EDMRST_GetTopLevelTemplates */
           RE_get_top_level_templates_result
               re_get_top_level_templates(
                                   RE_get_top_level_templates_args ) = 4;

           /* rpc for EDMRST_Submit */
           RE_status_result
               re_submit( RE_submit_args ) = 5;

           /* rpc for EDMRST_GetSubmitResults */
           RE_get_submit_results_output
               re_get_submit_results( RE_get_submit_results_args ) = 6;

           /* rpc for EDMRST_Start */
           RE_status_result
               re_start( RE_start_args ) = 7;

           /* rpc for EDMRST_GetRestoreFeedback */
           RE_get_restore_feedback_result
               re_get_restore_feedback( RE_get_restore_feedback_args ) = 8;

           /* rpc for EDMRST_GetQuestion */
           RE_get_question_result
               re_get_question( RE_null_args ) = 9;

           /* rpc for EDMRST_SetUserAnswer */
           RE_status_result
```



```
               re_set_user_answer( RE_set_user_answer_args ) = 10;

           /* rpc for EDMRST_Finish */
           RE_status_result
               re_finish( RE_null_args ) = 11;

           /* rpc for EDMRST_DoesAlternateExist */
           RE_boolean_result
               re_does_alternate_exist( RE_does_alternate_exist_args ) = 12;
           /* rpc's for EDMRST_GetRestorableObjects */
           RE_get_restorable_objects_start_result
               re_get_restorable_objects_start(
                                   RE_get_restorable_objects_start_args ) = 13;
           RE_get_restorable_objects_output_result
               re_get_restorable_objects_output(
                                   RE_get_restorable_objects_output_args ) = 14;

           /* rpc's for EDMRST_FindRestorableObjects */
           RE_find_restorable_objects_result
               re_find_restorable_objects(
                                   RE_find_restorable_objects_args ) = 15;
           RE_get_find_results_result
               re_get_find_results( RE_get_find_results_args ) = 16;

           /* rpc's for EDMRST_MarkObject */
           RE_mark_object_result
               re_mark_object( RE_mark_object_args ) = 17;
           RE_get_mark_results_result
               re_get_mark_results( RE_get_mark_results_args ) = 18;

           /* rpc's for EDMRST_UnmarkObject */
           RE_mark_object_result
               re_unmark_object( RE_unmark_object_args ) = 19;
           RE_get_unmark_results_result
               re_get_unmark_results( RE_get_mark_results_args ) = 20;

           /* rpc for EDMRST_GetMarkedTotalSize */
           RE_get_marked_total_size_result
               re_get_marked_total_size( RE_null_args ) = 21;

           /* rpc for EDMRST_GetCurrentTemplate */
           RE_get_current_template_result
               re_get_current_template( RE_null_args ) = 22;

           /* rpc for EDMRST_GetCurrentBackupTime */
           RE_get_current_backup_time_result
               re_get_current_backup_time( RE_null_args ) = 23;

           /* rpc for EDMRST_GetAllBackupTimes */
           RE_get_all_backup_times_result
               re_get_all_backup_times( RE_get_all_backup_times_args ) = 24;

           /* rpc for EDMRST_IsTherePrevBackup */
           RE_boolean_result
               re_is_there_prev_backup( RE_set_backup_time_args ) = 25;

           /* rpc for EDMRST_IsThereNextBackup */
           RE_boolean_result
               re_is_there_next_backup( RE_set_backup_time_args ) = 26;

           /* rpc for EDMRST_IsTherePrevBackupForTime */
           RE_boolean_result
               re_is_there_prev_backup_for_time(
                                   RE_backup_for_time_args) = 27;

           /* rpc for EDMRST_IsThereNextBackupForTime */
```

```
 RE_boolean_result
     re_is_there_next_backup_for_time(
                                    RE_backup_for_time_args) = 28;


 /* rpc for EDMRST_SetBackupForTime */
  RE_status_result
     re_set_backup_for_time( RE_backup_for_time_args ) = 29;

 /* rpc for EDMRST_SetPrevBackup */
  RE_status_result
     re_set_prev_backup( RE_set_backup_time_args ) = 30;

 /* rpc for EDMRST_SetNextBackup */
  RE_status_result
     re_set_next_backup( RE_set_backup_time_args ) = 31;

 /* rpc for EDMRST_SetFirstBackup */
  RE_status_result
     re_set_first_backup( RE_set_backup_time_args ) = 32;

 /* rpc for EDMRST_SetMostRecentBackup */
  RE_status_result
     re_set_most_recent_backup( RE_set_backup_time_args ) = 33;

 /* rpc for EDMRST_GetNecessaryMedia */
  RE_get_necessary_media_result
     re_get_necessary_media( RE_get_necessary_media_args ) = 34;

 /* rpc for EDMRST_IsObjectMarkable */
  RE_is_object_markable_result
     re_is_object_markable( RE_is_object_markable_args ) = 35;

 /* rpc for EDMRST_IsObjectMarked */
  RE_is_object_marked_result
     re_is_object_marked( RE_is_object_marked_args ) = 36;

 /* rpc for EDMRST_GetDestinationHosts */
 RE_get_hosts_result
     re_get_destination_hosts( RE_get_hosts_args ) = 37;

 /* rpc for EDMRST_GetHostPlatformType */
 RE_get_host_platform_type_result
     re_get_host_platform_type( RE_string_args ) = 38;

 /* rpc for EDMRST_IsObjectSearchable */
 RE_boolean_result
     re_is_object_searchable( RE_tlo_query_args ) = 39;

 /* rpc for EDMRST_GetBackupTimesSupport */
 RE_boolean_result
     re_get_backup_times_support( RE_tlo_query_args ) = 40;

 /* rpc for EDMRE_Load_recx_directives*/
 RE_status_result
     re_load_recx_directives( RE_recx_file_info ) = 41;

 /* rpc for EDMRST_poll_Load_recx_directives */
 RE_status_result
     re_poll_load_recx_directives(RE_null_args ) = 42;

 /* rpc for RSTSL_get_backup_level */
 RE_catalog_info
     re_get_catalog_info(RE_time) = 43;

 /* rpc for EDMRST_GetAllTopLevelObjects */
 RE_get_top_level_objects_result
```



```
     re_get_all_top_level_objects(
                                    RE_get_top_level_objects_args ) = 44;

 /* rpc for EDMRST_GetSymmRestoreOption */
 RE_boolean_result
     re_get_symm_restore_option( RE_tlo_query_args ) = 45;

 /* rpc for EDMRST_Ping */
 RE_status_result
     re_ping( RE_null_args ) = 46;


   ) = 1; /* This is version 1 */

%/* This is the RPC program number.
                                                    These are reserved in /pds/docs/RPC_numbers
% * This number cannot be re-used by any other RPC daemon on the machine,
                                                                        as it
% * identifies this daemon uniquely.  If it were to be re-used,
                                                    the last daemon
% * to register would be contacted when RPC's come in for this number.
%*/
} = 390016;
```

```
 /* -- Template created by NEURON DATA Open Interface.                */
 /* -- Do not alter 'CodeGen' directives.                             */
 /* {{ CodeGen: GeneratorVersion 4 }}                                 */

 /* -- Code generated on 05/20/97 at 16:10:09.                        */
 /* -- Code regenerated on 05/22/97 at 16:07:31.                      */
 /* -- Code regenerated on 05/22/97 at 16:09:45.                      */
 /* -- Code regenerated on 05/23/97 at 09:24:06.                      */
 /* -- Code regenerated on 05/23/97 at 09:34:04.                      */
 /* -- Code regenerated on 05/23/97 at 09:34:48.                      */
 /* -- Code regenerated on 09/22/97 at 12:51:42.                      */
 /* -- Code regenerated on 09/22/97 at 16:36:51.                      */
 /* -- Code regenerated on 03/18/98 at 11:14:28. */
 /* -- Code regenerated on 05/18/99 at 10:10:12.                      */
 /* -- Code regenerated on 05/18/99 at 10:54:55.                      */
 /* -- Code regenerated on 06/18/99 at 15:20:38.                      */
 /* -- Code regenerated on 07/15/99 at 11:19:43.                      */
 /* -- Code regenerated on 07/16/99 at 15:49:52.                      */
 /* -- Code regenerated on 09/03/99 at 10:48:06.                      */
 /* {{ CodeGen: CodeHistory }}                                        */

#define ERR_LIB RESTORE

#include <stdlib.h>
#include <libgen.h>

#include <esl/c_portable.h>

#define REST_INIT
#include "restoreP.h"
#include "restore.h"
#undef REST_INIT

#include <X11/Xlib.h>
#include <xpub.h>

#include "restore/restMgr.h"
#include "restSearch.h"
#include "restSelMgr.h"
#include "restCBMgr.h"
#include "restUtils.h"
#include "gutil/aboutMgr.h"
#include "gutil/alertMgr.h"
#include "gutil/grpMgr.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/hostUtils.h"
#include "gutil/winUtils.h"
#include "gutil/tedUtils.h"
#include "gutil/lboxUtils.h"
#include "gutil/cboxUtils.h"
#include "gutil/codeTracer.h"
#include "gutil/resUtils.h"
#include "help/helpIpc.h"
#include "ipc/ipcl.h"

ERR_EXTERN
ERR_INMODULE("restore")

#define  HELP_OPTION    "help"
#define  VERSION_OPTION "v"
#define  SYNC_OPTION    "sync"
```



```
 /* {{ CodeGen: ClassImplementationPlaceHolder }}                     */
 /* {{ CodeGen: WinClassImplementationPlaceHolder }}                  */

 /* {{ CodeGen: WindowSection RestoreWin                              */
 /* ================================================================ */
 /* == Code for Window "RestoreWin"                                  */
 /* ================================================================ */

 /* {{ CodeGen: MenuImplementationPlaceHolder }}                      */

 static  void  C_FAR  S_RestoreWinNfy L2(WinPtr, win, WinNfyEnum, code)
 {

   /* Call the restore notify routine so update the timeout status */
   REST_NotifyReceived (code);

   switch (code) {
   case WIN_NFYTERMINATE:
     if (REST_Remove ())
     {
       WIN_DefNfy(win, code);
       EVENT_MainExit ();
     }
     break;
   /* USER CODE */
   case WIN_NFYMOUSECLICK:
     GUTIL_WinHandleMouseClick (win);
     break;
   case WIN_NFYREDRAW:
     WIN_DefNfy(win, code);
     /*
      * Update the focus borders
      */

     if (WGT_HasFocus ((WgtPtr)REST_RestoreWin->BackupListBox, BOOL_TRUE))
       GUTIL_DrawFocusBorder ((
                               WgtPtr)REST_RestoreWin->BackupListBox, BOOL_TRUE);

     /* Use the junkTed to determine if the backup sarea has focus */
     if (WGT_HasFocus ((WgtPtr)REST_RestoreWin->JunkTEd, BOOL_TRUE))
       GUTIL_DrawFocusBorder ((WgtPtr)REST_RestoreWin->BackupSArea, BOOL_TRUE);

     if (WGT_HasFocus ((WgtPtr)REST_RestoreWin->SelectedListBox, BOOL_TRUE))
       GUTIL_DrawFocusBorder ((
                               WgtPtr)REST_RestoreWin->SelectedListBox, BOOL_TRUE);

     if (WGT_HasFocus ((WgtPtr)REST_RestoreWin->MediaListBox, BOOL_TRUE))
       GUTIL_DrawFocusBorder ((WgtPtr)REST_RestoreWin->MediaListBox, BOOL_TRUE);

     break;
   case WIN_NFYRESIZE:
     GUTIL_WinHandleResize ((WinPtr)win);
     break;

   default:
     WIN_DefNfy(win, code);
   }

 }

 static  void  C_FAR  S_MediaListBoxNfy L2(LBoxPtr, lbox, LBoxNfyEnum, code)
 {

   switch (code) {
   /* USER CODE */
   case LBOX_NFYCELLDELETE:
```

```
    REST_DisposeMediaInfo (lbox);
    break;
  default:
    GUTIL_LBOX_DefNfy (lbox, code, REST_GetMediaListColumnValues);
  }

}

static  void  C_FAR  S_SelectedListBoxNfy L2(LBoxPtr, lbox, LBoxNfyEnum, code)
{

  switch (code) {
  /* USER CODE */
  case LBOX_NFYCELLDELETE:
    REST_DisposeSelectedInfo (lbox);
    break;
  case LBOX_NFYSELOPERATION:
    REST_UpdateRemoveButtons ();
    break;
  default:
    GUTIL_LBOX_DefNfy(lbox, code, REST_GetSelectedListColumnValues);
  }

}

/* {{ CodeGen: WgtNfyHandler HitPrevBackupButton                          */
static   void C_FAR RestoreRestoreWin_HitPrevBackupButton L1(
                                                  RestoreRestoreWinPtr, win)
{
  REST_PrevButtonSelect ();
}
/* }} CodeGen: WgtNfyHandler HitPrevBackupButton                          */

/* {{ CodeGen: WgtNfyHandler HitCalendarButton                            */
static   void C_FAR RestoreRestoreWin_HitCalendarButton L1(
                                                  RestoreRestoreWinPtr, win)
{
  REST_CalendarButtonSelect ();
}
/* }} CodeGen: WgtNfyHandler HitCalendarButton                            */

/* {{ CodeGen: WgtNfyHandler HitNextBackupButton                          */
static   void C_FAR RestoreRestoreWin_HitNextBackupButton L1(
                                                  RestoreRestoreWinPtr, win)
{
  REST_NextButtonSelect ();
}
/* }} CodeGen: WgtNfyHandler HitNextBackupButton                          */

/* {{ CodeGen: WgtNfyHandler EltSelectedTemplateBox                       */
static   void C_FAR RestoreRestoreWin_EltSelectedTemplateBox L2(
                    RestoreRestoreWinPtr, win, CBoxEltSelectedNfyCPtr, info)
{
  REST_UpdateTemplateFromBoxes ();
}
/* }} CodeGen: WgtNfyHandler EltSelectedTemplateBox                       */

static BoolEnum S_TemplateSelectProc (Str selectedStr)
{
  REST_UpdateTemplateFromBoxes ();

  return (BOOL_TRUE);
}

static  void  C_FAR  S_TemplateBoxNfy L2(CBoxPtr, cbox, CBoxNfyEnum, code)
{
```



```
  GUTIL_CBOX_DefNfy (cbox, code, S_TemplateSelectProc);
}

/* {{ CodeGen: WgtNfyHandler EltSelectedPrimaryBox                        */
static   void C_FAR RestoreRestoreWin_EltSelectedPrimaryBox L2(
                    RestoreRestoreWinPtr, win, CBoxEltSelectedNfyCPtr, info)
{
  REST_UpdateTemplateFromBoxes ();
}
/* }} CodeGen: WgtNfyHandler EltSelectedPrimaryBox                        */

/* {{ CodeGen: WgtNfyHandler ValidateJunkTEd                              */
static   void C_FAR RestoreRestoreWin_ValidateJunkTEd L1(
                                                  RestoreRestoreWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler ValidateJunkTEd                              */

/* {{ CodeGen: WgtNfyHandler CellStringBackupListBox                      */
static   void C_FAR RestoreRestoreWin_CellStringBackupListBox L2(
                                  RestoreRestoreWinPtr, win, LBoxStringPtr, lbs)
{
}
/* }} CodeGen: WgtNfyHandler CellStringBackupListBox                      */

/* {{ CodeGen: WgtNfyHandler ValidateBackupListBox                        */
static   void C_FAR RestoreRestoreWin_ValidateBackupListBox L1(
                                                  RestoreRestoreWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler ValidateBackupListBox                        */

/* {{ CodeGen: WgtNfyHandler HitSearchButton                              */
static   void C_FAR RestoreRestoreWin_HitSearchButton L1(
                                                  RestoreRestoreWinPtr, win)
{
  REST_DisplaySearch ();
}
/* }} CodeGen: WgtNfyHandler HitSearchButton                              */

/* {{ CodeGen: WgtNfyHandler HitMarkButton                                */
static   void C_FAR RestoreRestoreWin_HitMarkButton L1(
                                                  RestoreRestoreWinPtr, win)
{
  REST_MarkBackupItems ();
}
/* }} CodeGen: WgtNfyHandler HitMarkButton                                */

/* {{ CodeGen: WgtNfyHandler HitUnmarkButton                              */
static   void C_FAR RestoreRestoreWin_HitUnmarkButton L1(
                                                  RestoreRestoreWinPtr, win)
{
  REST_UnmarkBackupItems ();
}
/* }} CodeGen: WgtNfyHandler HitUnmarkButton                              */

/* {{ CodeGen: WgtNfyHandler HitTypeSortButton                            */
static   void C_FAR RestoreRestoreWin_HitTypeSortButton L1(
                                                  RestoreRestoreWinPtr, win)
{
  REST_SetSort (REST_ByType);
}
/* }} CodeGen: WgtNfyHandler HitTypeSortButton                            */

/* {{ CodeGen: WgtNfyHandler HitNameSortButton                            */
static   void C_FAR RestoreRestoreWin_HitNameSortButton L1(
```

```
                                                    RestoreRestoreWinPtr, win)
 {
  REST_SetSort (REST_ByName);
 }
 /* )) CodeGen: WgtNfyHandler HitNameSortButton                          */

 /* (( CodeGen: WgtNfyHandler HitOwnerSortButton                         */
static   void C_FAR RestoreRestoreWin_HitOwnerSortButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  REST_SetSort (REST_ByOwner);
}
/* )) CodeGen: WgtNfyHandler HitOwnerSortButton                         */

/* (( CodeGen: WgtNfyHandler HitSizeSortButton                          */
static   void C_FAR RestoreRestoreWin_HitSizeSortButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  REST_SetSort (REST_BySize);
}
/* )) CodeGen: WgtNfyHandler HitSizeSortButton                          */

/* (( CodeGen: WgtNfyHandler HitDateSortButton                          */
static   void C_FAR RestoreRestoreWin_HitDateSortButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  REST_SetSort (REST_ByDate);
}
/* )) CodeGen: WgtNfyHandler HitDateSortButton                          */

/* (( CodeGen: WgtNfyHandler HitHiddenButton                            */
static   void C_FAR RestoreRestoreWin_HitHiddenButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  REST_ShowHiddenFiles = TBUT_GetSelected ((TButPtr)win->HiddenButton);
  REST_UpdateViewOptions ();
}
/* )) CodeGen: WgtNfyHandler HitHiddenButton                            */

/* (( CodeGen: WgtNfyHandler HitBadFilesButton                          */
static   void C_FAR RestoreRestoreWin_HitBadFilesButton L1(
                                                    RestoreRestoreWinPtr, win)
{
 REST_ShowBadFiles = TBUT_GetSelected ((TButPtr)win->BadFilesButton);
  REST_UpdateViewOptions ();
}
/* )) CodeGen: WgtNfyHandler HitBadFilesButton                          */

/* (( CodeGen: WgtNfyHandler HitMarkBadButton                           */
static   void C_FAR RestoreRestoreWin_HitMarkBadButton L1(
                                                    RestoreRestoreWinPtr, win)
{
 REST_MarkBadFiles = TBUT_GetSelected ((TButPtr)win->MarkBadButton);
}
/* )) CodeGen: WgtNfyHandler HitMarkBadButton                           */

/* (( CodeGen: WgtNfyHandler ValidateRestoreItemsTarea                  */
static   void C_FAR RestoreRestoreWin_ValidateRestoreItemsTarea L1(
                                                    RestoreRestoreWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateRestoreItemsTarea                  */

/* (( CodeGen: WgtNfyHandler ValidateRestoreSizeTarea                   */
static   void C_FAR RestoreRestoreWin_ValidateRestoreSizeTarea L1(
                                                    RestoreRestoreWinPtr, win)
```



```
{
}
/* )) CodeGen: WgtNfyHandler ValidateRestoreSizeTarea                   */

/* (( CodeGen: WgtNfyHandler ValidateBadFilesText                       */
static   void C_FAR RestoreRestoreWin_ValidateBadFilesText L1(
                                                    RestoreRestoreWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateBadFilesText                       */

/* (( CodeGen: WgtNfyHandler HitRemoveButton                            */
static   void C_FAR RestoreRestoreWin_HitRemoveButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  REST_RemoveSelectedItems ();
}
/* )) CodeGen: WgtNfyHandler HitRemoveButton                            */

/* (( CodeGen: WgtNfyHandler HitClearButton                             */
static   void C_FAR RestoreRestoreWin_HitClearButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  REST_RemoveAllSelectedItems ();
}
/* )) CodeGen: WgtNfyHandler HitClearButton                             */

/* (( CodeGen: WgtNfyHandler HitCloseButton                             */
static   void C_FAR RestoreRestoreWin_HitCloseButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  WIN_Terminate ((WinPtr)win);
}
/* )) CodeGen: WgtNfyHandler HitCloseButton                             */

/* (( CodeGen: WgtNfyHandler HitStartButton                             */
static   void C_FAR RestoreRestoreWin_HitStartButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  REST_StartRestoral ();
}
/* )) CodeGen: WgtNfyHandler HitStartButton                             */

static WgtPtr focusWgt;

static   void C_FAR RestoreRestoreWin_FocusHelpButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  focusWgt = PANEL_GetFocusWgt ((PanelPtr)WGT_GetWin((WgtPtr){
                                                          win->HelpButton}));
}

/* (( CodeGen: WgtNfyHandler HitHelpButton                              */
static   void C_FAR RestoreRestoreWin_HitHelpButton L1(
                                                    RestoreRestoreWinPtr, win)
{
  REST_DisplayContextHelp (focusWgt);
}
/* )) CodeGen: WgtNfyHandler HitHelpButton                              */

/* (( CodeGen: WgtNfyHandler ValidateBackupDateText                     */
static   void C_FAR RestoreRestoreWin_ValidateBackupDateText L1(
                                                    RestoreRestoreWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateBackupDateText                     */
```

```
/* (( CodeGen: WgtNfyHandler HitViewOptionsTab                    */
static   void C_FAR RestoreRestoreWin_HitViewOptionsTab L1(
                                                    RestoreRestoreWinPtr, win)

{
}
/* )) CodaGen: WgtNfyHandler HitViewOptionsTab                    */

/* (( CodeGen: WgtNfyHandler HitMarkSummaryTab                    */
static   void C_FAR RestoreRestoreWin_HitMarkSummaryTab L1(
                                                    RestoreRestoreWinPtr, win)

{
}
/* )) CodeGen: WgtNfyHandler HitMarkSummaryTab                    */

/* (( CodeGen: WgtNfyHandler CellStringSelectedListBox            */
static   void C_FAR RestoreRestoreWin_CellStringSelectedListBox L2(
                                RestoreRestoreWinPtr, win, LBoxStringPtr, lbs)

{
}
/* )) CodeGen: WgtNfyHandler CellStringSelectedListBox            */

/* (( CodeGen: WgtNfyHandler ValidateSelectedListBox              */
static   void C_FAR RestoreRestoreWin_ValidateSelectedListBox L1(
                                                    RestoreRestoreWinPtr, win)

{
}
/* )) CodeGen: WgtNfyHandler ValidateSelectedListBox              */

/* (( CodeGen: WgtNfyHandler HitMediaTab                          */
static   void C_FAR RestoreRestoreWin_HitMediaTab L1(
                                                    RestoreRestoreWinPtr, win)

{
}
/* )) CodeGen: WgtNfyHandler HitMediaTab                          */

/* (( CodeGen: WgtNfyHandler CellStringMediaListBox               */
static   void C_FAR RestoreRestoreWin_CellStringMediaListBox L2(
                                RestoreRestoreWinPtr, win, LBoxStringPtr, lbs)

{
}
/* )) CodeGen: WgtNfyHandler CellStringMediaListBox               */

/* (( CodeGen: WgtNfyHandler ValidateMediaListBox                 */
static   void C_FAR RestoreRestoreWin_ValidateMediaListBox L1(
                                                    RestoreRestoreWinPtr, win)

{
}
/* )) CodeGen: WgtNfyHandler ValidateMediaListBox                 */

/* (( CodeGen: WgtNfyHandler ValidatePathTEd                      */
static   void C_FAR RestoreRestoreWin_ValidatePathTEd L1(
                                                    RestoreRestoreWinPtr, win)

{
}
/* )) CodeGen: WgtNfyHandler ValidatePathTEd                      */

/* (( CodeGen: WgtNfyHandler HitAllowPartialButton                */
static   void C_FAR RestoreRestoreWin_HitAllowPartialButton L1(
                                                    RestoreRestoreWinPtr, win)

{
}
/* )) CodeGen: WgtNfyHandler HitAllowPartialButton                */
/* (( CodeGen: WgtNfyPlaceHolder ))                               */

static  void  C_FAR  S_PathTEdNfy L2(TEdPtr, ted, TEdNfyEnum, code)
```



```
{
  /* If this is a keyboard character, and it is a return, set the view */
  if ((code == TED_NFYKEYCHAR) &&
      ((EVENT_GetKeyCode() == EVENT_KEYRETURN) ||
       (EVENT_GetKeyCode() == EVENT_KEYENTER)))
  {
    REST_SetViewToPath ();
  }
  /* Else, let the utilities do their thing */
  else
  {
    GUTIL_TED_DefNfy (ted, code);
  }
}

/* (( CodeGen: UseDefaultNfyHandler name_of_nfy_handler ))        */
/* (( CodeGen: UseAllDefaultNfyHandlers name_of_wgt_member ))     */

void      RestoreRestoreWin_Construct L1(RestoreRestoreWinPtr, win)
{
          /* (( CodeGen: WgtInitializations                       */
          win->DataAvailablePanel = (PanelPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "DataAvailablePanel");
          win->FSOptionsPanel = (PanelPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "FSOptionsPanel");
          win->BackupDateTarea = (TAreaPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "BackupDateTarea");
          win->BackupDateText = (STEdPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "BackupDateText");
          win->PrevBackupButton = (PButPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "PrevBackupButton");
          win->CalendarButton = (PButPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "CalendarButton");
          win->NextBackupButton = (PButPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "NextBackupButton");
          win->TemplateBox = (CBoxPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "TemplateBox");
          win->PrimaryBox = (CBoxPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "PrimaryBox");
          win->JunkTEd = (STEdPtr)PANEL_GetNamedWgt((PanelPtr)win, "JunkTED");
          win->BackupSArea = (SAreaPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "BackupSArea");
          win->BackupListBox = (LBoxPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "BackupListBox");
          win->LboxVSb = (SbPtr)PANEL_GetNamedWgt((PanelPtr)win, "LboxVSb");
          win->PathTArea = (TAreaPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "PathTArea");
          win->PathTEd = (STEdPtr)PANEL_GetNamedWgt((PanelPtr)win, "PathTED");
          win->SearchButton = (PButPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "SearchButton");
          win->MarkButton = (PButPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "MarkButton");
          win->UnmarkButton = (PButPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "UnmarkButton");
          win->TabsPanel = (PanelPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "TabsPanel");
          win->MarkSummaryTab = (PButPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "MarkSummaryTab");
          win->MarkSummaryPanel = (PanelPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "MarkSummaryPanel");
          win->ItemsSelectedLabel = (TAreaPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "ItemsSelectedLabel");
          win->SelectedListBox = (LBoxPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "SelectedListBox");
          win->RestoreItemsLabel = (TAreaPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "RestoreItemsLabel");
```

```
win->RestoreItemsTarea = (STEdPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "RestoreItemsTarea");
win->RestoreItemsLabel1 = (TAreaPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "RestoreItemsLabel1");
win->RestoreSizeTarea = (STEdPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "RestoreSizeTarea");
win->BadFilesLabel = (TAreaPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "BadFilesLabel");
win->BadFilesText = (STEdPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "BadFilesText");
win->RemoveButton = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "RemoveButton");
win->ClearButton = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "ClearButton");
win->MediaTab = (PButPtr)PANEL_GetNamedWgt((PanelPtr)win, "MediaTab");
win->MediaPanel = (PanelPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "MediaPanel");
win->MediaListBox = (LBoxPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "MediaListBox");
win->ViewOptionsTab = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "ViewOptionsTab");
win->ViewOptionsPanel = (PanelPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "ViewOptionsPanel");
win->SortPanel = (PanelPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "SortPanel");
win->TypeSortButton = (RButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "TypeSortButton");
win->NameSortButton = (RButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "NameSortButton");
win->OwnerSortButton = (RButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "OwnerSortButton");
win->SizeSortButton = (RButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "SizeSortButton");
win->DateSortButton = (RButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "DateSortButton");
win->OptionsPanel = (PanelPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "OptionsPanel");
win->HiddenButton = (CButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "HiddenButton");
win->BadFilesButton = (CButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "BadFilesButton");
win->MarkBadButton = (CButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "MarkBadButton");
win->AllowPartialButton = (CButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "AllowPartialButton");
win->CloseButton = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "CloseButton");
win->StartButton = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "StartButton");
win->HelpButton = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "HelpButton");
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->BackupDateText, TED_NFYVALIDATE,
        (
      WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateBackupDateText);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->PrevBackupButton, TBUT_NFYHIT,
          (WinWgtNfyHandlerProc)RestoreRestoreWin_HitPrevBackupButton);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->CalendarButton, TBUT_NFYHIT,
          (WinWgtNfyHandlerProc)RestoreRestoreWin_HitCalendarButton);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->NextBackupButton, TBUT_NFYHIT,
          (WinWgtNfyHandlerProc)RestoreRestoreWin_HitNextBackupButton);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->TemplateBox, CBOX_NFYELTSELECTED,
```



```
        (
      WinWgtNfyHandlerProc)RestoreRestoreWin_EltSelectedTemplateBox);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->PrimaryBox, CBOX_NFYELTSELECTED,
        (
       WinWgtNfyHandlerProc)RestoreRestoreWin_EltSelectedPrimaryBox);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->JunkTEd, TED_NFYVALIDATE,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateJunkTEd);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->BackupListBox, LBOX_NFYCELLSTRING,
        (
     WinWgtNfyHandlerProc)RestoreRestoreWin_CellStringBackupListBox);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->BackupListBox, LBOX_NFYVALIDATE,
        (
      WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateBackupListBox);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->PathTEd, TED_NFYVALIDATE,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_ValidatePathTEd);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->SearchButton, TBUT_NFYHIT,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_HitSearchButton);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->MarkButton, TBUT_NFYHIT,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_HitMarkButton);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->UnmarkButton, TBUT_NFYHIT,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_HitUnmarkButton);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->MarkSummaryTab, TBUT_NFYHIT,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_HitMarkSummaryTab);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->SelectedListBox, LBOX_NFYCELLSTRING,
        (
     WinWgtNfyHandlerProc)RestoreRestoreWin_CellStringSelectedListBox);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->SelectedListBox, LBOX_NFYVALIDATE,
        (
      WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateSelectedListBox);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->RestoreItemsTarea, TED_NFYVALIDATE,
        (
    WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateRestoreItemsTarea);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->RestoreSizeTarea, TED_NFYVALIDATE,
        (
     WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateRestoreSizeTarea);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->BadFilesText, TED_NFYVALIDATE,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateBadFilesText);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->RemoveButton, TBUT_NFYHIT,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_HitRemoveButton);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->ClearButton, TBUT_NFYHIT,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_HitClearButton);
WIN_SetWgtNfyHandler((WinPtr)win, (WgtPtr)win->MediaTab, TBUT_NFYHIT,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_HitMediaTab);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->MediaListBox, LBOX_NFYCELLSTRING,
        (
     WinWgtNfyHandlerProc)RestoreRestoreWin_CellStringMediaListBox);
WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->MediaListBox, LBOX_NFYVALIDATE,
         (WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateMediaListBox);
```

```
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->ViewOptionsTab, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitViewOptionsTab);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->TypeSortButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitTypeSortButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->NameSortButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitNameSortButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->OwnerSortButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitOwnerSortButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->SizeSortButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitSizeSortButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->DateSortButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitDateSortButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->HiddenButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitHiddenButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->BadFilesButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitBadFilesButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->MarkBadButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitMarkBadButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->AllowPartialButton, TBUT_NFYHIT,
            (
          WinWgtNfyHandlerProc)RestoreRestoreWin_HitAllowPartialButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->CloseButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitCloseButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->StartButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitStartButton);
    WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->HelpButton, TBUT_NFYHIT,
            (WinWgtNfyHandlerProc)RestoreRestoreWin_HitHelpButton);
    /* )) CodeGen: WgtInitializations 808688                    */

    /*
     * Remove notifies for the TAB buttons, we have to leave them
     * above so that ND regenerates OK.
     */


    WIN_RemoveWgtNfyHandler((WinPtr)win,
                            (WgtPtr)win->ViewOptionsTab,
                            TBUT_NFYHIT);
    WIN_RemoveWgtNfyHandler((WinPtr)win,
                            (WgtPtr)win->MarkSummaryTab,
                            TBUT_NFYHIT);
    WIN_RemoveWgtNfyHandler((WinPtr)win,
                            (WgtPtr)win->MediaTab,
                            TBUT_NFYHIT);
    WIN_RemoveWgtNfyHandler((WinPtr)win,
                            (WgtPtr)win->PathTEd,
                           TED_NFYVALIDATE);

    WIN_RemoveWgtNfyHandler((WinPtr)win,
                           (WgtPtr)win->TemplateBox,
                           CBOX_NFYELTSELECTED);

    /*
     * Reset the utility widgets to the utility style notifies
```



```
     */
            WGT_SetNfyProc ((WgtPtr)win, S_RestoreWinNfy);
            WGT_SetNfyProc ((WgtPtr)win->BackupDateText, GUTIL_OutputTED_DefNfy);
            WGT_SetNfyProc((WgtPtr)win->SelectedListBox, S_SelectedListBoxNfy);
            WGT_SetNfyProc ((
                             WgtPtr)win->RestoreItemsTarea, GUTIL_OutputTED_DefNfy);
            WGT_SetNfyProc ((
                             WgtPtr)win->RestoreSizeTarea, GUTIL_OutputTED_DefNfy);
            WGT_SetNfyProc ((WgtPtr)win->BadFilesText, GUTIL_OutputTED_DefNfy);
            WGT_SetNfyProc((WgtPtr)win->MediaListBox, S_MediaListBoxNfy);
            WGT_SetNfyProc((WgtPtr)win->PathTEd, S_PathTEdNfy);
            WGT_SetNfyProc((WgtPtr)win->TemplateBox, S_TemplateBoxNfy);

    /* Setup the path widget to use the utilities */
    GTED_SetDefaults ((TEdPtr)win->PathTEd,
                      TED_Alpha,
                      GMAX_PATHNAME_LENGTH,
                      0,
                      0,
                      STR_STD_FILENAME);

            WIN_SetWgtNfyHandler((WinPtr)win, (
                                  WgtPtr)win->HelpButton, TBUT_NFYGAINFOCUS,
                    (WinWgtNfyHandlerProc)RestoreRestoreWin_FocusHelpButton);
}

void    REST_RestoreWinLoadInit LO()
{
            RestoreRestoreWinPtr win;

            (void)RLIB_LoadLibFile("restore", "restore.dat");

            win = (RestoreRestoreWinPtr)WIN_LoadSized("restore", "RestoreWin",
                        sizeof(RestoreRestoreWinRec));
            RestoreRestoreWin_Construct(win);

            REST_RestoreWin = win;

            WIN_Init((WinPtr)win);
}

static void REST_PrintUsageAndExit (Str command,
                                    Str invalidOption)
{
        /* Print out the bad option if one was given */
  if (invalidOption != NULL)
  {
    STR_Printf ("%s: bad command line option \"%s\"\n\n",
                command,
                invalidOption);
  }

  /* Print out the useage */
  STR_Printf (
                                                  "usage:\n\t%s [-options ...
 ]\n\nwhere options include:\n\t-help\t\t\tprint out this message\n\t-display
                                                  displayname\tX server to contact\n\n",
              command);

    fflush(stderr);
    fflush(stdout);
    _exit(0);
}

/* )) CodeGen: WindowSection RestoreWin                     */
```

```
/* (( CodeGen: WindowImplementationPlaceHolder )) */

/* (( CodeGen: MainSection )) */
/* ========================================================================== */
/* == Code for Main == */
/* ========================================================================== */

#include <nd.h>

ERR_DECLARE

/*
 *  "main()" entrypoint
 */

/*********************************************************************
 * main
 *
 * Description:
 *    This is the main routine, it will begin the restoral process.
 *
 * Parameters:
 *   argc (I) - The count of the command line arguements.
 *   argv (I) - The string list of the command line arguements.
 *
 * Returns:
 *   None.
 *
 *********************************************************************/

int      main L2(int, argc, char**, argv)
{
  int        i;                      /* Loop counter */
  BoolEnum   traceOn = BOOL_FALSE;   /* Flag if tracing should be turned on */
  char      *displayString;          /* String for DISPLAY env variable */
  int        nkey = 0;               /* Number of keys */
  int        key1;                   /* First Key */
  int        key2;                   /* Second Key */
  int        inputQ;                 /* Shared help queue for input */
  int        outputQ;                /* Shared help queue for output */
  BoolEnum   usingIPC = BOOL_FALSE;
                                     /* Flag if we are talking with main view */
  BoolEnum   sharedHelp = BOOL_FALSE; /* Flag if sharing help with main view */
  BoolEnum   synchronize = BOOL_FALSE;/* Flag if we shoul run in X sync mode */
  char      *message;                /* Message to send to main view */
  IPCLHandle handle;                 /* Handle to comms to main view */
  ipclStatus status;                 /* pass/fail status of ipc connection */
  BoolEnum   setColors = BOOL_FALSE; /* Flag if user specified color scheme */
  Str        colorStr;               /* Color schema string from args */
  Int        colorRet = RESOURCE_NULL_ARGC; /* Return val from color func */
  GALERT_WinHandle synchHandle = NULL;

  /* Register this program */
  GUTIL_RegisterProgram (argv[0]);

  /* Set the restore from client flag to false for starters */
  REST_RestoreFromClient = BOOL_FALSE;

  /* Initialize the global color value for color schemes */
  colorVal = argc + 1 ;
  colorArgs = argc;

  /* Initialize the variable REST_RestoreClient */
  STR_Cpy (REST_RestoreClient, "");

  /* Loop through the command line arguments */
```



```
  for (i=1; i<argc; i++)
  {
    /* Check if this is a HELP option */
    if (strcmp(argv[i], "-"HELP_OPTION) == 0)
    {
      REST_PrintUsageAndExit (basename(argv[0]), NULL);
    }

    /* Check if this is a version option */
    else if (strcmp(argv[i], "-"VERSION_OPTION) == 0)
    {
      GUTIL_PrintVersionAndExit ();
    }

    /* Check if this is the DISPLAY option */
    else if (strcmp(argv[i], "-" DISPLAY_OPTION) == 0)
    {
      /* Get the next argument and set the display variable to it */
      i++;
      GUTIL_SetDisplay (argv[i]);
    }

    /* Check if this is the Client side restoral option */
    else if (strcmp(argv[i], "-" FROM_CLIENT_OPTION) == 0)
    {

      /* Flag that this is client side restoral */
      REST_RestoreFromClient = BOOL_TRUE;

      /* Get the client name */
      i++;
      STR_Cpy (REST_RestoreClient, argv[i]);
    }

    /* Check for the -color option */
/*
 * COLOR SCHEME NOT SUPPORTED IN THIS RELEASE!!!!
 *

    else if (strncmp(argv[i],
                     "-" RES_COLOR_OPTION,
                     strlen(RES_COLOR_OPTION) + 1) == 0)
    {
      i++;
      setColors = BOOL_TRUE;
      colorStr = (Str) argv[i];
    }
 *
 */

    /* If this is the IPC Key option, get the keys */
    else if (strcmp(argv[i], "-" IPCl_KEY_OPTION) == 0)
    {
      nkey = atoi(argv[++i]);
      key1 = atoi(argv[++i]);
      if(nkey == 2)
        key2 = atoi(argv[++i]);
      else
        key2 = key1;
      usingIPC = BOOL_TRUE;
    }

    /* If this is the Help Queue option, get the queues */
    else if (strcmp(argv[i], "-" RESTORE_HELP_Q_OPT) == 0)
    {
      inputQ = atoi(argv[++i]);
```

```
      outputQ = atoi(argv[++i]);
      sharedHelp = BOOL_TRUE;
    }

    /* If this is the trace option */
    else if (strncmp(argv[i], "-" TRACE_OPTION, strlen(TRACE_OPTION)+1) == 0)
    {
      /* Turn on tracing */
      traceOn = BOOL_TRUE;
    }

    /* Check if this is the client list option */
    else if (strcmp(argv[i], "-" RESTORE_CLIENT_OPT) == 0)
    {
      /* skip all following arguments up to the next option */
      i++;
      while ((i<argc) && (argv[i][0] != '-'))
      {
        i++;
      }

      /* If we exitted the loop because of a new option,
                                                    rewind to the option */
      if (i<argc)
        i--;
    }

    /* Check if this is the work item list option */
    else if (strcmp(argv[i], "-" RESTORE_WI_OPT) == 0)
    {
      /* skip all following arguments up to the next option */
      i++;
      while ((i<argc) && (argv[i][0] != '-'))
      {
        i++;
      }

      /* If we exitted the loop because of a new option,
                                                    rewind to the option */
      if (i<argc)
        i--;
    }

    /* Check if this is the synchronize option */
    else if (strncmp(argv[i], "-"SYNC_OPTION, strlen(SYNC_OPTION) + 1) == 0)
    {
      synchronize = BOOL_TRUE;
    }

    else
    {
      REST_PrintUsageAndExit (basename(argv[0]), argv[i]);
    }
  }

  /* Check the initial Environment */
  GUTIL_CheckInitEnv (argv[0]);

  /* default initialization */
  ERR_MAININIT;
  ERR_MODULEUSE;

  /*
   * Initialize ND stuff
   */
```



```
  ND_Init (argc, argv);

  /* Initialize the utilities */
  GUTIL_Initialize ();

  /* Install the generic signal handlers */
  GUTIL_AddGenericSignalHandlers ((
                                  GUTIL_SignalHandlerProc) REST_SignalHandler);

  /* Initialize the use of RPCs in this process */
  GUTIL_InitializeRPC ();

  /* if there exists an argument after the -color */
  if (setColors)
    colorRet = GUTIL_ReadResFile (colorStr, BOOL_TRUE);

  /* if there wasn't an argument or if the read failed */
  if (colorRet != RESOURCE_FILE_OK)
    GUTIL_ReadResFile ("", BOOL_TRUE);

  /* Set the defaults to cover people who don't set correctly */
  GUTIL_SetResourcesToDefaults (BOOL_TRUE);

  /* Set the running directory */
  GUTIL_SetRunningDirectory (basename (argv[0]));

  /* Never, I mean NEVER, let the user see OpenLook (yuk!) */
  if (DSPLY_GetLook () == DSPLY_LOOKOPENLOOK)
    DSPLY_SetLook (DSPLY_LOOKMOTIF);

  /* Initialize the restore components */
  REST_Initialize ();

  /* Show the about box while initializing if not from edm */
  if (!usingIPC)
  {
    ABOUT_DisplayBanner (NULL);
    EVENT_Update ();
  }
  /* Otherwise, show an initializing message */
  else
  {
    synchHandle = GALERT_DisplaySynchronousWait (NULL,
                                    REST_GetErrorString (REST_INIT_TITLE),
                                    GICON_GetIcon (I_WAIT),
                                    REST_GetErrorString (REST_INIT_MESSAGE),
                                    BOOL_FALSE);

    EVENT_Update ();
  }

  /*
   * Determine the list of clients to display in the file manager
   */

  REST_StartClientList ();
  for (i=1; i<argc; i++)
  {
    /* Check if this is an option */
    if (argv[i][0] == '-')
    {
      /* If this is the client list option */
      if (strcmp(argv[i], "-" RESTORE_CLIENT_OPT) == 0)
      {

        /* Get all following arguments up to the next option */
        i++;
```

```
        while ((i<argc) && (argv[i][0] != '-'))
        {
          /* This is another client to add to the list */
          REST_AddClient (argv[i]);
          i++;
        }

        /* OK, we can back up one */
        i--;
      }

    }
  }

  /*
   * Create the list of selected work items
   */

  REST_StartWiList ();
  for (i=1; i<argc; i++)
  {
    /* Check if this is an option */
    if (argv[i][0] == '-')
    {
      /* If this is the client list option */
      if (strcmp(argv[i], "-" RESTORE_WI_OPT) == 0)
      {

        /* Get all following arguments up to the next option */
        i++;
        while ((i<argc) && (argv[i][0] != '-'))
        {
          /* This is another work item to add to the list */
          REST_AddWi (argv[i]);
          i++;
        }

        /* OK, we can back up one */
        i--;
      }

    }
  }

  /* If syncronous mode, set mode */
  if (synchronize)
  {
    XSynchronize (X_Display(), BOOL_TRUE);
  }

  /* IF tracing is on */
  if (traceOn)
  {
    /* start with everything */
    TRACESETFLAGS(V_TRACE_EVERYTHING)

    /* start tracing */
    TRACESTART
    TRACESTARTFILEIO

    /* Display the trace controls window */
    TRACEPOPUPCONTROLS
  }

  /* If we are talking with mainview, tell it we're up */
  if (usingIPC)
```



```
  {
    status = ipclOpen (&handle, key1, key2);
    if ((handle != NULL) && (status != IPCL_FAILURE))
    {
      message = (char *) GUTIL_Malloc (strlen(IPC_CONNECT_STRING) + 1);
      STR_Cpy (message, IPC_CONNECT_STRING);
      ipclSendMessage (handle,
                       IPCL_NOWAIT,
                       1,
                       message,
                       strlen(IPC_CONNECT_STRING) + 1);
      GUTIL_Free (message);
    }
  }

  /* Display the restore window */
  REST_Display ();

  /* If we are sharing help with mainview, talk to the already running help */
  if (sharedHelp)
  {
    EDMHELP_InitQueues (inputQ, outputQ);
  }

  /* Remove the about box */
  if (!usingIPC)
  {
    ABOUT_TerminateWin ();
    EVENT_Update ();
  }
  else if (synchHandle != NULL)
  {
    GALERT_CancelSynchDialog (synchHandle);
    EVENT_Update ();
  }

  /* Put up the window */
  WIN_Show((WinPtr)REST_RestoreWin);

  /* Begin the event processing */
  ND_Run();

  /*
   * OK, we're outta here!  Clean up and go home
   */

  /* Close help */
  EDMHELP_End ();

  /* default termination */
  ND_Exit ();

  /* Exit the process */
  return EXIT_OK;
}
/* }} CodeGen: MainSection                                          */
/* {{ CodeGen: MainPlaceHolder }}                                   */
```

```
/*****************************************************************************
 * restMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for the EDM Restore window.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *      $RCSfile$
 *      $Revision$
 *      $Date$
 *****************************************************************************/


#define  ERR_LIB  RESTORE

#include <esl/c_portable.h>

#include <stdlib.h>

#include <libgen.h>
#include <time.h>

#include <apppub.h>
#include <eventpub.h>
#include <rlibpub.h>
#include <gwpub.h>
#include <respub.h>
#include <lboxpub.h>
#include <mbarpub.h>
#include <panelpub.h>
#include <tareapub.h>
#include <tbutpub.h>
#include <tedpub.h>
#include <winpub.h>
#include <strlpub.h>
#include <drawpub.h>
#include <dsplypub.h>
#include <ibutpub.h>
#include <arraypub.h>

#include "eerrno/e_errno.h"
#include "util/esl_string.h"
#include <restore/restore_api.h>
#define REST_MGR_INIT
#include "restore/restMgr.h"
#undef REST_MGR_INIT
#include "restore.h"
#include "restoreP.h"
#include "restCalendar.h"
#include "restSearch.h"
#include "restSelMgr.h"
#include "restFileMgr.h"
#include "restUtils.h"
#include "restCBMgr.h"
#include "restAPIUtils.h"
#include "gutil/timeUtils.h"
```



```
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/resUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/fileMgr.h"
#include "gutil/tabDefs.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/hostUtils.h"
#include "gutil/cboxUtils.h"
#include "gutil/codeTracer.h"
#include "gutil/alertMgr.h"
#include "help/helpDefs.h"
#include "help/helpIpc.h"
#include "ipc/ipcl.h"

ERR_EXTERN
ERR_INMODULE("restore")

/*************
 * Constants *
 *************/

#define MAX_TIMES_BUFFER       64
#define REST_MARK_THRESHOLD  1000
#define CONNECT_TIMEOUT_SEC    60

/*************************
 * Local Data Structures *
 *************************/

/*************************
 * Local Global Variables *
 *************************/

/* Current list of clients */
static RestoreClientPtr currentClientList = NULL;

/* Current list of work items */
static RestoreWorkItemPtr currentWIList = NULL;

/* Flags for current state of processing */
static BoolEnum  updatingDate = BOOL_FALSE;

/* Static variables for mark progress */
static GALERT_WinHandle synchMarkHandle = NULL;

/* Handle to the current fill in progress dialog */
static GALERT_WinHandle synchFillHandle = NULL;

/* Forward declaration of signal handler */
void REST_SignalHandler (int sig);

/* Forward declaration of validate work item routine */
static BoolEnum REST_ValidateWorkItem (GREST_Object  workItemObject,
                                       eerrno_ty     *errorCode);

static BoolEnum GREST_IsObjectMarked (serverHandle serverHandle,
                                      GREST_Object object)
{
  unsigned long numChecked;
  boolean_ty    isMarked = BOOL_FALSE;

  if ((serverHandle != NULL) && (object != NULL))
  {
```

```
    EDMRST_IsObjectMarked (serverHandle,
                           1,
                           &object,
                           &numChecked,
                           &isMarked);
  }

  return (isMarked);
}

/*****************************************************************
 * REST_DisplayBackupDate
 *
 * Description:
 *   This routine will display the current backup date and set the
 *   date buttons to their correct states.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void  REST_DisplayBackupDate (void)
{
  Char      dateString[GMAX_OBJECT_LENGTH]; /* Date to display */
  time_t    thisTime;                       /* Current backup time */
  eerrno_ty eerrno;                         /* Error status */

  /* Get the current backup time */
  if ((eerrno = EDMRST_GetCurrentBackupTime(
                                        GREST_Handle, &thisTime)) == E_SUCCESS)
  {

    /* Print the time and date to the date string */
    STR_Sprintf (dateString, "%s", REST_GetTimeDateString (thisTime));

  }
  else
  {
    /* Couldn't get a backup time, must not have a work item yet */
    STR_Sprintf (dateString, "");
    thisTime = 0;
  }

  /* Set the date text label */
  TED_SetStr ((TEdPtr)REST_RestoreWin->BackupDateText, dateString);

  /* Update the sensitivity of the date buttons */
  REST_UpdateDateButtons ();
}

/*****************************************************************
 * REST_UpdateBackupDate
 *
 * Description:
 *   This routine will update the current work item and the displayed
 *   date after a date change.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
```



```
 *
 *****************************************************************/

void  REST_UpdateBackupDate (void)
{

  /* Flag that we are updating the backup date */
  updatingDate = BOOL_TRUE;

  /* Re-Read the current work item */
  REST_ReReadWorkItem (currentWorkItemInfo);

  /* Display the new backup date */
  REST_DisplayBackupDate ();

  /* Flag that we are no longer updating the backup date */
  updatingDate = BOOL_FALSE;
}

/*****************************************************************
 * REST_UpdateBackupTemplates
 *
 * Description:
 *   This routine will update the template and trailset boxes
 *   to the current choices available.
 *
 * Parameters:
 *   info (I) - Work-Item info record to get the templates for
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void  REST_UpdateBackupTemplates (RestoreInfoPtr info)
{
  template_name_ty     currentTemplate;
                                                    /* Current template */
  template_name_ty     templates[TEMPLATE_BUFFER_LENGTH]; /* New templates */
  BoolEnum             isAlternate;                 /* Alternate flag */
  BoolEnum             exists = BOOL_FALSE;         /* Exists flag */
  BoolEnum             currentFound = BOOL_FALSE;   /* Found flag */
  long                 cookie = INIT_COOKIE;
                                                    /* The magic cookie */
  short                numEntries;                  /* Number found */
  Int                  i;                           /* Loop counter */
  eerrno_ty            eerrno;                      /* Error status */

  /* Validate the object passed in */
  if ((info != NULL) &&
      (info->type == REST_WorkItem) &&
      (info->restoreObject != NULL))
  {

    /* First, clear out any old Templates: */
    CBOX_GoFirst(REST_RestoreWin->TemplateBox);
    while (CBOX_IsOk (REST_RestoreWin->TemplateBox))
    {
      CBOX_GoFirst (REST_RestoreWin->TemplateBox);
      CBOX_CurRemoveElt (REST_RestoreWin->TemplateBox);
    }

    /* get the current template */
    if (EDMRST_GetCurrentTemplate (GREST_Handle,
                                   currentTemplate,
                                   &isAlternate) != E_SUCCESS)
```

```
    {
      STR_Cpy (currentTemplate, "");
      isAlternate = BOOL_FALSE;
    }

    /* Get all the templates for the new work item */
    while (cookie != DONE_COOKIE)
    {
      numEntries = 0;
      if ((eerrno = EDMRST_GetTopLevelTemplates (GREST_Handle,
                                                 info->restoreObject,
                                                 TEMPLATE_BUFFER_LENGTH,
                                                 templates,
                                                 &numEntries,
                                                 &cookie)) == E_SUCCESS)
      {

        /* Add each entry to the template box */
        CBOX_GoFirst(REST_RestoreWin->TemplateBox);
        for (i=0; i<numEntries; i++)
        {
          CBOX_CurAddElt (REST_RestoreWin->TemplateBox, (ClientPtr)NULL);
          CBOX_CurSetLabel (REST_RestoreWin->TemplateBox, templates[i]);

          /* If this is the current template, select it */
          if (STR_Cmp (templates[i], currentTemplate) == CMP_EQUAL)
          {
            REST_SelectCurrentTemplate ();
            currentFound = BOOL_TRUE;
          }
          CBOX_GoNext (REST_RestoreWin->TemplateBox);
        }

        if ((!currentFound) && (STR_Cmp (currentTemplate, "") != CMP_EQUAL))
        {
          CBOX_CurAddElt (REST_RestoreWin->TemplateBox, (ClientPtr)NULL);
          CBOX_CurSetLabel (REST_RestoreWin->TemplateBox, currentTemplate);
          REST_SelectCurrentTemplate ();
          numEntries++;
          currentFound = BOOL_TRUE;
        }

        /* If we didn't find it (probably no savesets) select the first */
        if ((!currentFound) && (numEntries > 0))
        {
          CBOX_GoFirst (REST_RestoreWin->TemplateBox);
          REST_SelectCurrentTemplate ();
        }
      }
      else
      {
        /* At least add the current template */
        if (STR_Cmp (currentTemplate, "") != CMP_EQUAL)
        {
          CBOX_CurAddElt (REST_RestoreWin->TemplateBox, (ClientPtr)NULL);
          CBOX_CurSetLabel (REST_RestoreWin->TemplateBox, currentTemplate);
          REST_SelectCurrentTemplate ();
        }

        /* Just get out */
        cookie = DONE_COOKIE;
      }
    }

    /* First, clear out any old trails: */
    CBOX_GoFirst(REST_RestoreWin->PrimaryBox);
```



```
    while (CBOX_IsOk (REST_RestoreWin->PrimaryBox))
    {
      CBOX_GoFirst (REST_RestoreWin->PrimaryBox);
      CBOX_CurRemoveElt (REST_RestoreWin->PrimaryBox);
    }

    /* Add the primary trail (always exists) */
    CBOX_GoFirst(REST_RestoreWin->PrimaryBox);
    CBOX_CurAddElt (REST_RestoreWin->PrimaryBox, (ClientPtr)NULL);
    CBOX_CurSetLabel (REST_RestoreWin->PrimaryBox,
                      (Str)STRL_GetNthStr(REST_TrailNameList,
                                          REST_PRIMARY_TRAIL_INDEX));
    CBOX_CurSetId (REST_RestoreWin->PrimaryBox, 1);

    /* Add the alternate trail if it exists */
    EDMRST_DoesAlternateExist (GREST_Handle,
                               info->restoreObject,
                               currentTemplate,
                               &exists);
    if (exists)
    {
      CBOX_GoNext (REST_RestoreWin->PrimaryBox);
      CBOX_CurAddElt (REST_RestoreWin->PrimaryBox, (ClientPtr)NULL);
      CBOX_CurSetLabel (REST_RestoreWin->PrimaryBox,
                        (Str)STRL_GetNthStr(REST_TrailNameList,
                                            REST_ALTERNATE_TRAIL_INDEX));
      CBOX_CurSetId (REST_RestoreWin->PrimaryBox, 2);
    }

    /* Show whether or not it is the alternate */
    CBOX_GoFirst (REST_RestoreWin->PrimaryBox);
    if (isAlternate)
    {
      CBOX_GoNext (REST_RestoreWin->PrimaryBox);
    }
    REST_SelectCurrentTrail ();

  }

}

/*****************************************************************
 * REST_UpdateChildMarks
 *
 * Description:
 *   This routine will update the marked flag for all children and
 *   recursively for their children for the given object.
 *
 * Parameters:
 *   parentObject (I) - The object whose children need to be updated
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_UpdateChildMarks (RestoreInfoPtr parentObject)
{
  RestoreInfoPtr tmpInfo;   /* Info to walk the list with */
  unsigned long  numChecked;

  /* Validate the input */
  if (parentObject != NULL)
  {
    /* Walk the children list */
    tmpInfo = parentObject->children;
```

```
    while (tmpInfo != NULL)
    {

      /* Determine if this object is marked */
      if (tmpInfo->restoreObject != NULL)
      {
        tmpInfo->marked = GREST_IsObjectMarked (GREST_Handle,
                                                tmpInfo->restoreObject);
      }

      /* Update the marks for the children of this object */
      REST_UpdateChildMarks (tmpInfo);

      /* Go to the next child */
      tmpInfo = tmpInfo->next;
    }
  }
}

/*********************************************************************
 * REST_UpdateObjectMarks
 *
 * Description:
 *   This routine will update the marked flag the given object and
 *   all its children.
 *
 * Parameters:
 *   parentObject (I) - The top level object whose children need to be updated
 *
 * Returns:
 *   None.
 *
 *********************************************************************/

void REST_UpdateObjectMarks (RestoreInfoPtr parentObject)
{
  RestoreInfoPtr nextChild;             /* Info to walk the list with */
  BoolEnum       allMarked = BOOL_TRUE; /* Flag if all children are marked */

  /* Validate the input */
  if (parentObject != NULL)
  {

    /* Determine if this object is marked */
    if (parentObject->restoreObject != NULL)
    {
      parentObject->marked = GREST_IsObjectMarked (GREST_Handle,

                                                   parentObject->restoreObject);
    }

    /* Update the marks for the children of this object */
    REST_UpdateChildMarks (parentObject);
  }

  /* See if the entire workitem is marked */
  if (currentWorkItemInfo != NULL)
  {
    nextChild = currentWorkItemInfo->children;
    while (allMarked && (nextChild != NULL))
    {
      if (!nextChild->marked)
      {
        allMarked = BOOL_FALSE;
      }
      else
```



```
      {
        nextChild = nextChild->next;
      }
    }
    currentWorkItemInfo->marked = allMarked;
  }
}

/*********************************************************************
 * REST_ClearChildMarks
 *
 * Description:
 *   This routine will clear the marked flag for all children and
 *   recursively for their children for the given object.
 *
 * Parameters:
 *   parentObject (I) - The object whose children need to be cleared
 *
 * Returns:
 *   None.
 *
 *********************************************************************/

void REST_ClearChildMarks (RestoreInfoPtr parentObject)
{
  RestoreInfoPtr tmpInfo;  /* Info to walk the list with */

  /* Validate the input */
  if (parentObject != NULL)
  {

    /* Walk the children list */
    tmpInfo = parentObject->children;
    while (tmpInfo != NULL)
    {

      /* Clear the marked flag for this object */
      if (tmpInfo->marked)
      {
        tmpInfo->marked = BOOL_FALSE;
        GFMGR_UpdateObject (REST_GetFMgrContext(), (GFMGR_Object)tmpInfo);
      }

      /* If this object is in the selected list, remove it */
      if (REST_IsItemSelected (tmpInfo) &&
          (tmpInfo->restoreObject != NULL))
      {
        REST_DeselectInfo (tmpInfo->restoreObject, 0);
      }

      /* Clear the marks for the children of this object */
      REST_ClearChildMarks (tmpInfo);

      /* Go to the next child */
      tmpInfo = tmpInfo->next;
    }
  }
}

/*********************************************************************
 * REST_ClearObjectMarks
 *
 * Description:
 *   This routine will clear the marked flag for the given object
 *   and for all descendants of the given object.
 *
```

```
 * Parameters:
 *   parentObject (I) - The object whose children need to be cleared
 *
 * Returns:
 *   None.
 *
 ********************************************************************/

void REST_ClearObjectMarks (RestoreInfoPtr parentObject)
{

  /* Validate the input */
  if (parentObject != NULL)
  {

    /* Clear the marked flag for this object */
    if (parentObject->marked)
    {
      parentObject->marked = BOOL_FALSE;
      GFMGR_UpdateObject (REST_GetFMgrContext(), (GFMGR_Object)parentObject);
    }

    /* If this object is in the selected list, remove it */
    if (REST_IsItemSelected (parentObject) &&
        (parentObject->restoreObject != NULL))
    {
        REST_DeselectInfo (parentObject, 0);
    }

    /* Clear the marks for the children of this object */
    REST_ClearChildMarks (parentObject);

  }

  /* Clear out any selection data */
  REST_ClearMarkedInfo ();

}

/********************************************************************
 * REST_GetCurrentClientInfo
 *
 * Description:
 *   This routine will return the current client object.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   The current cliennt object, or NULL if none.
 *
 ********************************************************************/

RestoreInfoPtr REST_GetCurrentClientInfo (void)
{
  RestoreInfoPtr returnInfo;  /* Info to return */

  /* If we are currently working with a work item, return it's parent */
  if (currentWorkItemInfo != NULL)
    returnInfo = currentWorkItemInfo->parent;

  /* Else, there is no current client */
  else
    returnInfo = NULL;
```



```
  /* Return the determined info */
  return (returnInfo);
}

/********************************************************************
 * REST_GetCurrentWorkItem
 *
 * Description:
 *   This routine will return the current work-item restorable object.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   The current work item object, or NULL if none.
 *
 ********************************************************************/

GREST_Object REST_GetCurrentWorkItem (void)
{
  GREST_Object returnObject;  /* The work item object to return */

  /* If we are currently looking at a work-item */
  if (currentWorkItemInfo != NULL)
    /* return the current work item object */
    returnObject = currentWorkItemInfo->restoreObject;
  else
    /* return NULL */
    returnObject = NULL;

  return (returnObject);
}

/********************************************************************
 * REST_CreateClientInfo
 *
 * Description:
 *   This routine will create the data for the given client.
 *
 * Parameters:
 *   clientName (I) - The name of the client
 *
 * Returns:
 *   The allocated data for the given client.
 *
 ********************************************************************/

RestoreInfoPtr  REST_CreateClientInfo (Str clientName)
{
  RestoreInfoPtr      newInfo;
                                                  /* Info created for client */
  BuCfgPlatformType  platformType = BUCFG_PLATFORM_UNKNOWN;
                                                  /* Platform type */

  /* Create a new info record */
  newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
  newInfo->parent = NULL;
  newInfo->children = NULL;
  newInfo->next = NULL;

  /* Copy in the fields */
  newInfo->name = esl_strdup (clientName);
  newInfo->type = REST_Client;

  /*
   * Determine the platform type and get the appropriate icon
```

```
    */

   if (EDMRST_GetHostPlatformType (
                          GREST_Handle, clientName, &platformType) == E_SUCCESS)
   {
     switch (platformType)
     {
       case BUCFG_PLATFORM_UNIX:
         newInfo->icon = GICON_GetIconBySize (I_UNIXCLIENT, ICON_SMALL);
         break;
       case BUCFG_PLATFORM_OS2:
         newInfo->icon = GICON_GetIconBySize (I_OS2CLIENT, ICON_SMALL);
         break;
       case BUCFG_PLATFORM_NETWARE:
         newInfo->icon = GICON_GetIconBySize (I_NETWARECLIENT, ICON_SMALL);
         break;
       case BUCFG_PLATFORM_WNT:
         newInfo->icon = GICON_GetIconBySize (I_WINNTCLIENT, ICON_SMALL);
         break;
       case BUCFG_PLATFORM_VMS:
         newInfo->icon = GICON_GetIconBySize (I_VMSCLIENT, ICON_SMALL);
         break;
       default:
         newInfo->icon = GICON_GetIconBySize (I_UNKNOWNCLIENT, ICON_SMALL);
     }
   }
   else
   {
     newInfo->icon = GICON_GetIconBySize (I_UNKNOWNCLIENT, ICON_SMALL);
   }

   newInfo->opened = BOOL_FALSE;
   newInfo->restoreObject = NULL;
   newInfo->status = Backup_Good;
   newInfo->backupTime = 0;
   newInfo->errorString = NULL;

   return (newInfo);
}

/*****************************************************************
 * REST_CreateErrorInfo
 *
 * Description:
 *   This routine will create the data for an error object
 *
 * Parameters:
 *   errString (I) - The error string for the new object
 *   parent    (I) - The parent of the error object
 *
 * Returns:
 *   The allocated data for the error object
 *
 *****************************************************************/

RestoreInfoPtr  REST_CreateErrorInfo (Str             errString,
                                      RestoreInfoPtr parent)
{
  RestoreInfoPtr      newInfo;    /* New info created for the error Object */

  /* Create the new object */
  newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
  newInfo->parent = parent;
  newInfo->children = NULL;
  newInfo->next = NULL;
```



```
  /* Fill in the data fields */
  if (errString != NULL)
    newInfo->name = esl_strdup (errString);
  else
    newInfo->name = NULL;
  newInfo->type = REST_ErrorObject;
  newInfo->opened = BOOL_FALSE;
  newInfo->marked = BOOL_FALSE;
  newInfo->restoreObject = NULL;
  newInfo->status = Backup_Good;
  newInfo->backupTime = 0;
  newInfo->errorString = NULL;

  newInfo->icon = REST_FailedIcon;

  /* Return the new object */
  return (newInfo);
}

/*****************************************************************
 * REST_CreateWorkItemInfo
 *
 * Description:
 *   This routine will create the data for the give work item object.
 *
 * Parameters:
 *   object (I) - The work item object.
 *   parent (I) - The parent of the new object
 *
 * Returns:
 *   The allocated data for the given work item.
 *
 *****************************************************************/

RestoreInfoPtr REST_CreateWorkItemInfo (GREST_Object   object,
                                        RestoreInfoPtr parent)
{
  RestoreInfoPtr      newInfo;    /* New info created for the WI Object */
  char                wiType;     /* Type of work item */
  eerrno_ty           eerrno;     /* Error code for failed workitem */

  /* Create the new object */
  newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
  newInfo->parent = parent;
  newInfo->children = NULL;
  newInfo->next = NULL;

  /* Fill in the data fields */
  newInfo->name = esl_strdup (EDMRST_GetObjectFullName (GREST_Handle, object));
  newInfo->type = REST_WorkItem;
  newInfo->opened = BOOL_FALSE;
  newInfo->marked = BOOL_FALSE;
  newInfo->restoreObject = object;
  newInfo->backupTime = 0;
  newInfo->errorString = NULL;

  /* Determine which icon to use */
  wiType = EDMRST_GetWorkItemType (GREST_Handle, object);
  if (wiType == WI_TYPE_OFFLINEDB)
  {
    newInfo->icon = REST_DBWorkItemIcon;
  }
  else
  {
    newInfo->icon = REST_FSWorkItemIcon;
  }
```

```
  if (!REST_ValidateWorkItem (object, &eerrno))
  {
    newInfo->type = REST_FailedWorkItem;
    newInfo->icon = REST_FailedWorkItemIcon;
    newInfo->errorString = esl_strdup (e_get_error_text(eerrno));
    newInfo->children = REST_CreateErrorInfo (newInfo->errorString, newInfo);
  }

  newInfo->status = Backup_Good;

  /* Return the new object */
  return (newInfo);
}

/*****************************************************************
 * REST_CreateDirectoryInfo
 *
 * Description:
 *   This routine will create the data for the given directory object.
 *
 * Parameters:
 *   object (I) - The directory object.
 *   parent (I) - The parent of the new object
 *
 * Returns:
 *   The allocated data for the given directory.
 *
 *****************************************************************/

RestoreInfoPtr REST_CreateDirectoryInfo (GREST_Object   object,
                                         RestoreInfoPtr parent)
{
  RestoreInfoPtr   newInfo;   /* New info created for the Object */

  /* Create the new object */
  newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
  newInfo->parent = parent;
  newInfo->children = NULL;
  newInfo->next = NULL;

  /* Fill in the data fields */
  newInfo->name = esl_strdup (EDMRST_GetObjectBaseName (GREST_Handle, object));
  newInfo->type = REST_Directory;
  newInfo->icon = REST_DirClosedIcon;
  newInfo->opened = BOOL_FALSE;
  newInfo->marked = GREST_IsObjectMarked (GREST_Handle, object);
  newInfo->restoreObject = object;
  newInfo->status = EDMRST_GetObjectStatus (GREST_Handle, object);
  newInfo->backupTime = 0;
  newInfo->errorString = NULL;

  /* Return the new object */
  return (newInfo);
}

/*****************************************************************
 * REST_CreateFileInfo
 *
 * Description:
 *   This routine will create the data for the given File object.
 *
 * Parameters:
 *   object (I) - The File object.
 *   parent (I) - The parent of the new object
 *
```



```
 * Returns:
 *   The allocated data for the given File.
 *
 *****************************************************************/

RestoreInfoPtr REST_CreateFileInfo (GREST_Object   object,
                                    RestoreInfoPtr parent)
{
  RestoreInfoPtr   newInfo;   /* New info created for the Object */

  /* Create the new object */
  newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
  newInfo->parent = parent;
  newInfo->children = NULL;
  newInfo->next = NULL;

  /* Fill in the data fields */
  newInfo->name = esl_strdup (EDMRST_GetObjectBaseName (GREST_Handle, object));
  newInfo->type = REST_File;
  newInfo->icon = REST_FileIcon;
  newInfo->opened = BOOL_FALSE;
  newInfo->restoreObject = object;
  newInfo->status = EDMRST_GetObjectStatus (GREST_Handle, object);
  newInfo->marked = GREST_IsObjectMarked (GREST_Handle, object);
  newInfo->backupTime = 0;
  newInfo->errorString = NULL;

  /* Return the new object */
  return (newInfo);
}

/*****************************************************************
 * REST_AddChild
 *
 * Description:
 *   This routine will add a child to a parent in a non-sorted order.
 *
 * Parameters:
 *   parent (I) - The parent object
 *   child  (I) - The new child object
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_AddChild (RestoreInfoPtr parent,
                    RestoreInfoPtr child)
{
  RestoreInfoPtr tmpInfo;             /* Pointer used to walk the children */

  if ((parent != NULL) && (child != NULL))
  {
    /* Add the new child to the list */
    child->next = parent->children;
    parent->children = child;
  }
}

/*****************************************************************
 * REST_CopyInfo
 *
 * Description:
 *   This routine copy the info from the source object to the destination.
 *   If the source info is the current work-item, update the current
 *   work-item to the destination.
```

```
 *
 * Parameters:
 *   sourceInfo (I)      - The object to copy from
 *   destinationInfo (I) - The object to copy to
 *
 * Returns:
 *   None.
 *
 ***********************************************************************/

void REST_CopyInfo (RestoreInfoPtr sourceInfo,
                    RestoreInfoPtr destinationInfo)
{

  if ((sourceInfo != NULL) && (destinationInfo != NULL))
  {

    /* NULL out the links */
    destinationInfo->parent = NULL;
    destinationInfo->children = NULL;
    destinationInfo->next = NULL;

    /* Copy each data field */
    destinationInfo->name           = esl_strdup (sourceInfo->name);
    destinationInfo->type           = sourceInfo->type;
    destinationInfo->icon           = sourceInfo->icon;
    destinationInfo->opened         = sourceInfo->opened;
    destinationInfo->restoreObject  = sourceInfo->restoreObject;
    destinationInfo->status         = sourceInfo->status;
    destinationInfo->backupTime     = sourceInfo->backupTime;

    /* If this was the current WI, set it to the new version */
    if (currentWorkItemInfo == sourceInfo)
      currentWorkItemInfo = destinationInfo;
  }
}

/***********************************************************************
 * REST_InitWorkItem
 *
 * Description:
 *   This routine will initialize a work-item.  This can be used so that
 *   the Restore API will consider the work-item the current work-item
 *   and the work-item routines can be called.
 *
 * Parameters:
 *   workItemObject (I) - The work-item object to initialize.
 *   errorCode      (O) - The error code received if we can't init.
 *
 * Returns:
 *   BOOL_TRUE  - If we successfully inited the work-item.
 *   BOOL_FALSE - If we unsuccessfully inited the work-item.
 *
 ***********************************************************************/

BoolEnum REST_InitWorkItem (GREST_Object  workItemObject,
                            eerrno_ty     *errorCode)
{
  GREST_Object objects[1];            /* Bogus buffer to get objects */
  long         cookie = INIT_COOKIE; /* Ah, the magic cookie */
  long         numEntries = 0;       /* Bogus value for object count */
  BoolEnum     init = BOOL_FALSE;    /* Flag if the init was sucessful */

  if (workItemObject != NULL)
  {
```



```
    /* Create one object */
    if (EDMRST_AllocRestorableObjects (GREST_Handle, objects, 1) == E_SUCCESS)
    {
      /* Attempt to get the object */
      if ((*errorCode = EDMRST_GetRestorableObjects (GREST_Handle,
                                                     workItemObject,
                                                     BOOL_TRUE,
                                                     1,
                                                     objects,
                                                     &numEntries,
                                                     &cookie)) == E_SUCCESS)
      {
        /* Successfully inited the work-item */
        init = BOOL_TRUE;
      }

      /* Free up the object */
      EDMRST_FreeRestorableObjects (GREST_Handle, objects, 1);
    }
  }

  /* Return whether or not we successfully inited the work-item */
  return (init);
}

/***********************************************************************
 * REST_GetMostRecentWITime
 *
 * Description:
 *   This routine will get the most recent backup time for the given
 *   work-item in the given template with the given trail.
 *
 * Parameters:
 *   workItemObject (I) - The work-item object to get the time for.
 *   template       (I) - The template to use.
 *   isAlternate    (I) - Flag whether or not to use the alternate trail
 *
 * Returns:
 *   The time of the most recent backup, 0 if no backups exist.
 *
 ***********************************************************************/

static time_t REST_GetMostRecentWITime (GREST_Object          workItemObject,
                                        template_name_ty      template,
                                        BoolEnum              isAlternate)
{
  BoolEnum      exists;       /* Flag if this template/trail exists */
  eerrno_ty     eerrno;       /* Error Status */
  time_t        thisTime = 0; /* Most recent time for work-item */
  u_long        flags;

  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
  else
    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

  /* If this is the alternate trail */
  if (isAlternate)
  {
    /* Check if there is an alternate trail for this template */
    EDMRST_DoesAlternateExist (
                               GREST_Handle, workItemObject, template, &exists);
  }
  else
  {
    /* Primary trails always exist */
```

```
    exists = BOOL_TRUE;
  }

  /* Set to this template using the given trailset */
  if ((exists) && (EDMRST_SetTopLevelTemplate (GREST_Handle,
                                               workItemObject,
                                               template,
                                               isAlternate) == E_SUCCESS))
  {
    /* Initialize the work-item */
    if (REST_InitWorkItem (workItemObject, &eerrno))
    {
      /* Get the most recent backup */
      if (EDMRST_SetMostRecentBackup (GREST_Handle, flags) == E_SUCCESS)
      {
        /* Initialize the work-item for this time */
        if (REST_InitWorkItem (workItemObject, &eerrno))
        {
          /* Get the time for this backup */
          if (EDMRST_GetCurrentBackupTime(
                                        GREST_Handle, &thisTime) != E_SUCCESS)
          {
            thisTime = 0;
          }
        }
      }
    }
  }

  return (thisTime);
}

/*****************************************************************
 * REST_GetMostRecentWI
 *
 * Description:
 *   This routine will set the template and trail for the given work-item
 *   to the most recent.
 *
 * Parameters:
 *   workItemObject (I) - The work-item object to initialize.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

static void REST_GetMostRecentWI (GREST_Object workItemObject)
{
  long              cookie = INIT_COOKIE;/* Ah, the magic cookie */
  short             numTemplates;        /* Number of templates returned */
  template_name_ty  origTemplate;        /* Original template to use */
  BoolEnum          origIsAlternate;     /* Original alternate flag to use */
  BoolEnum          origValid = BOOL_FALSE;/* Flag if original is valid */
  template_name_ty  currentTemplate;     /* Current template to use */
  BoolEnum          currentIsAlternate;  /* Current alternate flag to use */
  time_t            thisTime;            /* Current time for work-item */
  time_t            mostRecentTime = 0;  /* Most recent time found so far */
  template_name_ty  templates[TEMPLATE_BUFFER_LENGTH];
                                         /* Templates returned */
  Int               i;                   /* Loop Counter */
  eerrno_ty         eerrno;              /* Error code */
```



```
  u_long            flags;

  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
  else
    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

  if (REST_InitWorkItem (workItemObject, &eerrno))
  {
    if (EDMRST_GetCurrentTemplate (GREST_Handle,
                                   origTemplate,
                                   &origIsAlternate) == E_SUCCESS)
      origValid = BOOL_TRUE;
  }

  /* Keep looping while more templates exist */
  while (cookie != DONE_COOKIE)
  {
    /* Get the next group of templates */
    numTemplates = 0;
    if (EDMRST_GetTopLevelTemplates (GREST_Handle,
                                     workItemObject,
                                     TEMPLATE_BUFFER_LENGTH,
                                     templates,
                                     &numTemplates,
                                     &cookie) == E_SUCCESS)
    {
      /* Loop through each template */
      for (i=0; i<numTemplates; i++)
      {
        /* Get the most recent time for the primary trail */
        thisTime = REST_GetMostRecentWITime (workItemObject,
                                             templates[i],
                                             BOOL_FALSE);

        /* If this time is more recent than what we have so far */
        if (thisTime > mostRecentTime)
        {
          /* Mark this as the most recent backup */
          mostRecentTime = thisTime;
          STR_Cpy (currentTemplate, templates[i]);
          currentIsAlternate = BOOL_FALSE;
        }

        /* Get the most recent time for the alternate trail */
        thisTime = REST_GetMostRecentWITime (workItemObject,
                                             templates[i],
                                             BOOL_TRUE);

        /* If this time is more recent than what we have so far */
        if (thisTime > mostRecentTime)
        {
          /* Mark this as the most recent backup */
          mostRecentTime = thisTime;
          STR_Cpy (currentTemplate, templates[i]);
          currentIsAlternate = BOOL_TRUE;
        }
      }
    }
    else
    {
      /* Just get out */
```

```
      cookie = DONE_COOKIE;
    }
  }

  /* If we don't have a most recent time and the original was valid */
  if ((mostRecentTime == 0) && origValid)
  {
    /* Get the most recent time for the original template/trail */
    mostRecentTime = REST_GetMostRecentWITime (workItemObject,
                                               origTemplate,
                                               origIsAlternate);

    /* Set the current to the original */
    STR_Cpy (currentTemplate, origTemplate);
    currentIsAlternate = origIsAlternate;
  }

  /* If we found any templates */
  if (mostRecentTime != 0)
  {
    /* Set to the most recent template and trail */
    if (EDMRST_SetTopLevelTemplate (GREST_Handle,
                                    workItemObject,
                                    currentTemplate,
                                    currentIsAlternate) == E_SUCCESS)
    {
      /* Initialize this work-item */
      if (REST_InitWorkItem (workItemObject, &eerrno))
      {
        /* Set to the most recent time */
        EDMRST_SetBackupForTime (GREST_Handle, mostRecentTime, flags);
      }
    }
  }

}

/*****************************************************************
 * REST_ValidateWorkItem
 *
 * Description:
 *   This routine will determine if there are any valid backups for the
 *   given workitem.
 *
 * Parameters:
 *   workItemObject (I) - The work-item object to initialize.
 *   errorCode      (O) - The error code received if we can't init.
 *
 * Returns:
 *   BOOL_TRUE  - If we found a valid workitem
 *   BOOL_FALSE - If we did not find a valid workitem
 *
 *****************************************************************/

static BoolEnum REST_ValidateWorkItem (GREST_Object  workItemObject,
                                       eerrno_ty     *errorCode)

{
  long                cookie = INIT_COOKIE;   /* Ah, the magic cookie */
  short               numTemplates;
                                              /* Number of templates returned */
  template_name_ty    templates[TEMPLATE_BUFFER_LENGTH];
                                              /* Templates returned */
  Int                 i;                      /* Loop Counter */
  eerrno_ty           eerrno;                 /* Error code */
  BoolEnum            exists;                 /* Flag if alternate exists */
```



```
  BoolEnum            returnVal = BOOL_FALSE;
                                              /* Flag if we found a valid one */

  /* Try a simple init */
  if (REST_InitWorkItem (workItemObject, errorCode))
  {
    /* got one, wer'e done */
    returnVal = BOOL_TRUE;
  }

  /* Until we find a valid template,
                                              Keep looping while more templates exist */
  while ((returnVal == BOOL_FALSE) && (cookie != DONE_COOKIE))
  {
    /* Get the next group of templates */
    numTemplates = 0;
    if (EDMRST_GetTopLevelTemplates (GREST_Handle,
                                     workItemObject,
                                     TEMPLATE_BUFFER_LENGTH,
                                     templates,
                                     &numTemplates,
                                     &cookie) == E_SUCCESS)
    {
      /* Loop through each template */
      for (i=0; (i<numTemplates) && (returnVal == BOOL_FALSE); i++)
      {
        /* Set to this template using the given trailset */
        if (EDMRST_SetTopLevelTemplate (GREST_Handle,
                                        workItemObject,
                                        templates[i],
                                        BOOL_FALSE) == E_SUCCESS)
        {
          returnVal = REST_InitWorkItem (workItemObject, &eerrno);
        }

        if (returnVal == BOOL_FALSE)
        {
          /* Check if there is an alternate trail for this template */
          EDMRST_DoesAlternateExist (GREST_Handle,
                                     workItemObject,
                                     templates[i],
                                     &exists);

          if (exists)
          {
            if (EDMRST_SetTopLevelTemplate (GREST_Handle,
                                            workItemObject,
                                            templates[i],
                                            BOOL_TRUE) == E_SUCCESS)
            {
              returnVal = REST_InitWorkItem (workItemObject, &eerrno);
            }
          }
        }
      }
    }
    else
    {
      /* Just get out */
      cookie = DONE_COOKIE;
    }
  }

  return (returnVal);
```

```
}

/*****************************************************************
 * REST_AddWorkItems
 *
 * Description:
 *   This routine will add the work-items for the given parent.
 *
 * Parameters:
 *   parent (I) - The parent object to add work-items for
 *
 * Returns:
 *   BOOL_TRUE  - If any work-items were added.
 *   BOOL_FALSE - If no work-items were added.
 *
 *****************************************************************/

BoolEnum REST_AddWorkItems (RestoreInfoPtr parent)
{
  RestoreInfoPtr       info;                /* New WI info */
  BoolEnum             returnValue = BOOL_FALSE;  /* Flag if WIs exist */
  GREST_Object         workItems[WORK_ITEM_BUFFER_LENGTH];/* Array of WIs */
  GREST_Object        *unused;              /* Unused WIs in array */
  long                 cookie = INIT_COOKIE; /* Ah, the magic cookie */
  short                numEntries = 0;      /* Number of WIs found */
  int                  i;                   /* Loop Counter */
  char                 wiType;              /* Type of the work-item */
  eerrno_ty            eerrno;              /* Error Status */

  if (parent != NULL)
  {

    /* Get all the work items for the given client */
    while (cookie != DONE_COOKIE)
    {
      /* Allocate space for the next group of work-itmes */
      if (EDMRST_AllocRestorableObjects (GREST_Handle,
                                         workItems,
                                         WORK_ITEM_BUFFER_LENGTH) ==
                                                                  E_SUCCESS)
      {

        /* Get the work items */
        numEntries = 0;
        unused = workItems;
        if ((eerrno = EDMRST_GetTopLevelObjects (GREST_Handle,
                                                 parent->name,
                                                 WORK_ITEM_BUFFER_LENGTH,
                                                 workItems,
                                                 &numEntries,
                                                 &cookie)) == E_SUCCESS)
        {
          /* Loop through the returned work items */
          for (i=0; i<numEntries; i++)
          {

            /* Get the work-item type */
            wiType = EDMRST_GetWorkItemType (GREST_Handle, workItems[i]);

            /* If this is a listener work-item, don't show the user */
            if ((wiType ==  WI_TYPE_OLDB_KICKER) ||
                (wiType == WI_TYPE_OLDB_LISTENER))
            {
              /* We don't do any restorals on kicker or listener work items */
              EDMRST_FreeRestorableObjects (GREST_Handle, &(workItems[i]), 1);
            }
```



```
            /* Else this must be your everyday work-item */
            else
            {
              /* Create the work-item object */
              info = REST_CreateWorkItemInfo (workItems[i], parent);

              /* Add the work-item object to the children list */
              REST_AddChild (parent, info); /* Name of the work-item */
            }

            /* Bump the unused pointer past this one */
            unused++;
          }
        }
        else
        {
          /* We got an error, ignore it and get out */
          cookie = DONE_COOKIE;
        }

        /* Free up the left overs */
        if (numEntries < WORK_ITEM_BUFFER_LENGTH)
        {
          EDMRST_FreeRestorableObjects (GREST_Handle,
                                        unused,
                                        WORK_ITEM_BUFFER_LENGTH - numEntries);
        }
      }
      else
      {
        /* We got an error, ignore it and get out */
        cookie = DONE_COOKIE;
      }
    }

    /* Check if we added any work-items */
    returnValue = (returnValue);

    /* Now that we have all the children, sort them */
    REST_SortChildren (parent);
  }

  /* Return if we added any children */
  return (returnValue);
}

/*****************************************************************
 * REST_ValidateClient
 *
 * Description:
 *   This routine will verify a host has something that can be restored
 *
 * Parameters:
 *   clientName (I) - The name of the client to validate
 *
 * Returns:
 *   BOOL_TRUE  - If any work-items can be restored
 *   BOOL_FALSE - If no work-items can be restored
 *
 *****************************************************************/

BoolEnum REST_ValidateClient (Str clientName)
{
  BoolEnum             isValid = BOOL_FALSE; /* Flag if WIs exist */
  GREST_Object         workItems[WORK_ITEM_BUFFER_LENGTH];/* Array of WIs */
```

```
  long                cookie = INIT_COOKIE; /* Ah, the magic cookie */
  short               numEntries;           /* Number of WIs found */
  Int                 i;                    /* Loop Counter */
  char                wiType;               /* Type of the work-item */
  eerrno_ty           eerrno;               /* Error code */

if (clientName != NULL)
{
  /* Keep getting workitems for the client until we find a valid one */
  while ((cookie != DONE_COOKIE) && (isValid == BOOL_FALSE))
  {
    /* Allocate space for the next group of work-itmes */
    if (EDMRST_AllocRestorableObjects (GREST_Handle,
                                       workItems,
                                       WORK_ITEM_BUFFER_LENGTH) ==
                                                              E_SUCCESS)
    {
      /* Get the work items */
      numEntries = 0;
      if (EDMRST_GetTopLevelObjects (GREST_Handle,
                                     clientName,
                                     WORK_ITEM_BUFFER_LENGTH,
                                     workItems,
                                     &numEntries,
                                     &cookie) == E_SUCCESS)
      {
        /* Loop through the returned work items */
        for (i=0; (i<numEntries) && (isValid == BOOL_FALSE); i++)
        {
          /* Get the work-item type */
          wiType = EDMRST_GetWorkItemType (GREST_Handle, workItems[i]);

          /* If this is not kicker or listener work-item, validate it */
          if ((wiType != WI_TYPE_OLDB_KICKER) &&
              (wiType != WI_TYPE_OLDB_LISTENER))
          {
            /* Attempt to init the workitem, if successful we found one */
            if (REST_ValidateWorkItem (workItems[i], &eerrno))
            {
              isValid = BOOL_TRUE;
            }
          }
        }
      }
      else
      {
        /* We got an error, ignore it and get out */
        cookie = DONE_COOKIE;
      }

      /* Free up the objects */
      EDMRST_FreeRestorableObjects (GREST_Handle,
                                    workItems,
                                    WORK_ITEM_BUFFER_LENGTH);
    }
    else
    {
      /* We got an error, ignore it and get out */
      cookie = DONE_COOKIE;
    }
  }
}
```



```
  /* Return whether or not we found something restorable */
  return (isValid);
}

/*****************************************************************
 * REST_CheckForFillCancel
 *
 * Description:
 *   This routine will determine if the user has cancelled the fill and
 *   will update the progress window.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   BOOL_TRUE  - If the user has cancelled
 *   BOOL_FALSE - otherwise
 *
 *****************************************************************/

static BoolEnum REST_CheckForFillCancel (Int totalItems)
{
  Char          outputString[MAX_STRING_LENGTH]; /* Updated string to display */
  BoolEnum      retVal;                          /* Value to return */

  if ((synchFillHandle == NULL) && (totalItems >= STATUS_REPORT_COUNT))
  {
    /* Build the new string */
    STR_Sprintf (outputString,
                 REST_GetErrorString (REST_FILL_STATUS),
                 totalItems);
    /* Initialize the window */
    synchFillHandle = GALERT_DisplaySynchronousWait
                        ((WinPtr)REST_RestoreWin,
                        REST_GetErrorString (REST_FILL_PROGRESS_TITLE),
                        GICON_GetIcon(I_WAIT),
                        outputString,
                        BOOL_TRUE);
    EVENT_Update ();
  }

  /* If the number of entries found has changed, update the string */
  if ((totalItems > 0) && ((totalItems % STATUS_REPORT_COUNT) == 0))
  {
    /* Build the new string */
    STR_Sprintf (outputString,
                 REST_GetErrorString (REST_FILL_STATUS),
                 totalItems);

    /* Update the progress dialog */
    GALERT_UpdateMessage (synchFillHandle, outputString);

  }

  /* Return whether or not the user cancelled */
  if (synchFillHandle != NULL)
    retVal = GALERT_IsCancelled(synchFillHandle);
  else
    retVal = BOOL_FALSE;

  return (retVal);
}

/*****************************************************************
 * REST_AddRestorableObjects
```

```
 *
 * Description:
 *   This routine will add restorable objects for the given parent object.
 *
 * Parameters:
 *   parent (I) - The parent object to get the children of.
 *
 * Returns:
 *   BOOL_TRUE  - If any children were added.
 *   BOOL_FALSE - If no children were added.
 *
 ***********************************************************************/

BoolEnum REST_AddRestorableObjects (RestoreInfoPtr parent)
{
  RestoreInfoPtr  info;                   /* New child info */
  BoolEnum        returnValue = BOOL_FALSE; /* Value to return */
  GREST_Object    objects[OBJECTS_BUFFER_LENGTH];  /* Objects returned */
  GREST_Object   *unused;                 /* Pointer to unused objects */
  long            cookie = INIT_COOKIE;   /* Ah, the magic cookie */
  long            numEntries;             /* Number of objects returned */
  Int             i;                      /* Loop counter */
  Int             totalCount = 0;
                                          /* The total number found so far */
  eerrno_ty       eerrno;                 /* Error status */

  if (parent != NULL)
  {

    /* If we are not updating the date, get the most rescent work-item */
    if ((parent->type == REST_WorkItem) && (!updatingDate))
    {
      REST_GetMostRecentWI (parent->restoreObject);
    }

    /* Initialize the fill handle */
    synchFillHandle = NULL;

    /* Get all the work items for the given client */
    while (!REST_CheckForFillCancel (totalCount) && (cookie != DONE_COOKIE))
    {

      /* Create the next group of objects */
      if (EDMRST_AllocRestorableObjects (GREST_Handle,
                                         objects,
                                         OBJECTS_BUFFER_LENGTH) == E_SUCCESS)
      {

        /* Retrieve the next group of objects */
        numEntries = 0;
        unused = objects;
        if ((eerrno = EDMRST_GetRestorableObjects (GREST_Handle,
                                                   parent->restoreObject,
                                                   BOOL_TRUE,
                                                   OBJECTS_BUFFER_LENGTH,
                                                   objects,
                                                   &numEntries,
                                                   &cookie)) == E_SUCCESS)
        {

          /* Loop through all objects */
          for (i=0; i<numEntries; i++)
          {

            /* If this is a directory create the directory object */
            if (EDMRST_IsObjectContainer (GREST_Handle, objects[i]))
```



```
            {
              info = REST_CreateDirectoryInfo (objects[i], parent);
            }

            /* If this is a file create the directory object */
            else if (EDMRST_IsObjectLeaf (GREST_Handle, objects[i]))
            {
              info = REST_CreateFileInfo (objects[i], parent);
            }

            /* Else this is an unknown object, treat it like a file */
            else
            {
              info = REST_CreateFileInfo (objects[i], parent);
            }

            /* Add this child to the parent */
            REST_AddChild (parent, info);
            totalCount++;
            returnValue = BOOL_TRUE;

            /* Bump the unused pointer past this one */
            unused++;
          }
        }
        else
        {
          /*
           * If this is not a re-read, display an error
           *   (else it would be displayed twice)
           */

          if (!REST_IsReReadInProgress())
          {
            Char outputString[MAX_STRING_LENGTH];  /* String to display */

            /* Get the message to display */
            STR_Sprintf (outputString,
                         REST_GetErrorString (REST_OPEN_OBJECT_ERROR),
                         EDMRST_GetObjectFullName (
                                        GREST_Handle, parent->restoreObject));

            /* display the error message */
            REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
                                      NULL,
                                      outputString,
                                      eerrno);
          }

          /* Get out of the loop */
          cookie = DONE_COOKIE;

          /*
           * If this was a workitem, flag it as failed and get out
           */

          if (parent->type == REST_WorkItem)
          {
            parent->type = REST_FailedWorkItem;
            parent->errorString = esl_strdup (e_get_error_text(eerrno));
            parent->children = REST_CreateErrorInfo (parent->errorString,
                                                     parent);
          }
        }

        /* Free up the left overs */
```

```
        if (numEntries < OBJECTS_BUFFER_LENGTH)
        {
          EDMRST_FreeRestorableObjects (GREST_Handle,
                                        unused,
                                        OBJECTS_BUFFER_LENGTH - numEntries);
        }
      }
      else
      {
        /* Got an error, ignore it and get out */
        cookie = DONE_COOKIE;
      }
    }

    /* The fill handle is not NULL, remove the progress window. */
    if (synchFillHandle != NULL)
    {
      GALERT_CancelSynchDialog (synchFillHandle);
      synchFillHandle = NULL;
    }

    /* Now that we have all the children, sort them */
    REST_SortChildren (parent);
  }

  /* Return whether or not we found children */
  return (returnValue);
}

/*****************************************************************
 * REST_CreateInfoChildren
 *
 * Description:
 *   This routine will create the children for the given object.
 *
 * Parameters:
 *   parent (I) - The parent object to get the children of.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_CreateInfoChildren (RestoreInfoPtr parent)
{

  if (parent != NULL)
  {

    /* Call the correct add routine based on type */
    switch (parent->type)
    {
      case REST_Client:
        REST_AddWorkItems (parent);
        break;
      case REST_WorkItem:
        REST_AddRestorableObjects (parent);
        break;
      case REST_Directory:
        REST_AddRestorableObjects (parent);
        break;
      case REST_File:
        break;
      default:
        break;
    }
```



```
  }
}

/*****************************************************************
 * REST_FindInfoInChildren
 *
 * Description:
 *   This routine will find the info object for the given full
 *   child name.
 *
 * Parameters:
 *   itemFullName (I) - The full name of the object to be found.
 *   parent       (I) - The parent object to start the search from.
 *   getChildren  (I) - Flag if new children should be created if necessary
 *
 * Returns:
 *   The found object or NULL if not found
 *
 *****************************************************************/

RestoreInfoPtr REST_FindInfoInChildren (Str             itemFullName,
                                        RestoreInfoPtr parent,
                                        BoolEnum       getChildren)
{
  RestoreInfoPtr tmpInfo;          /* Pointer to walk the children with */
  RestoreInfoPtr foundInfo = NULL; /* Matching info found */

  REST_StandardizePath(itemFullName);

  if (parent != NULL)
  {

    /* If there are no children yet, add them (but don't display) */
    if (getChildren && (parent->children == NULL) && (
                                        parent->type != REST_File))
    {
      /* Set the flag so that the objects aren't displayed */
      updatingDate = BOOL_TRUE;

      /* Add the child objects */
      REST_CreateInfoChildren (parent);

      /* Set the flag back */
      updatingDate = BOOL_FALSE;
    }

    /* Loop through the children */
    tmpInfo = parent->children;
    while ((foundInfo == NULL) && (tmpInfo != NULL))
    {

      /* Check if this is the one */
      if (STR_Cmp (itemFullName, REST_GetFullName (tmpInfo)) == CMP_EQUAL)
      {
        foundInfo = tmpInfo;
      }

      /* If this could be the parent, the check its children */
      else if ((tmpInfo->type != REST_File) &&
               (REST_IsParentString (tmpInfo, itemFullName)))
      {
        foundInfo = REST_FindInfoInChildren(
                                        itemFullName, tmpInfo, getChildren);
      }

      /* Go to the next child */
```

```
      tmpInfo = tmpInfo->next;
    }
  }

  /* Return the found object or NULL if not found */
  return (foundInfo);
}

/*****************************************************************
 * REST_StartClientList
 *
 * Description:
 *   This routine will initialize the global client list.  It will free up
 *   the old client list if it existed.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_StartClientList (void)
{
  RestoreClientPtr currClient;  /* Current client pointer to walk the list */
  RestoreClientPtr nextClient;  /* Next client in the list */

  /* If there is already a list, free it up */
  if (currentClientList != NULL)
  {
    /* Start at the beginning of the list */
    currClient = currentClientList;

    /* Loop until there are no more clients */
    while (currClient != NULL)
    {
      /* Save the next client pointer */
      nextClient = currClient->next;

      /* Free up this client */
      GUTIL_Free ((VoidPtr)currClient);

      /* Go to the next client */
      currClient = nextClient;
    }

    /* Set the current Client list to NULL */
    currentClientList = NULL;

  }

}

/*****************************************************************
 * REST_AddClient
 *
 * Description:
 *   This routine will add a client to the global client list.
 *
 * Parameters:
 *   clientName (I) - name of the client to add
 *
 * Returns:
 *   None.
 *
```



```
 *****************************************************************/

void REST_AddClient (Str clientName)
{
  RestoreClientPtr newClient;  /* The new client info */
  RestoreClientPtr nextClient; /* Pointer to walk the current list with */

  /* Create the new client */
  newClient = (RestoreClientPtr) GUTIL_Malloc (sizeof(RestoreClientRec));
  STR_Cpy (newClient->clientName, clientName);
  newClient->next = NULL;

  /* attach the new client to the end of the list */
  if (currentClientList != NULL)
  {
    nextClient = currentClientList;
    while (nextClient->next != NULL)
    {
      nextClient = nextClient->next;
    }
    nextClient->next = newClient;
  }
  else
  {
    currentClientList = newClient;
  }
}

/*****************************************************************
 * REST_StartWiList
 *
 * Description:
 *   This routine will initialize the global work item list.  It will free up
 *   the old work item list if it existed.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_StartWiList (void)
{
  RestoreWorkItemPtr currWI;  /* Current WI pointer to walk the list */
  RestoreWorkItemPtr nextWI;  /* Next WI in the list */

  /* If there is already a list, free it up */
  if (currentWIList != NULL)
  {
    /* Start at the beginning of the list */
    currWI = currentWIList;

    /* Loop until there are no more WIs */
    while (currWI != NULL)
    {
      /* Save the next WI pointer */
      nextWI = currWI->next;

      /* Free up this WI */
      GUTIL_Free ((VoidPtr)currWI);

      /* Go to the next WI */
      currWI = nextWI;
    }
```

```
    /* Set the current WI list to NULL */
    currentWIList = NULL;

  }

}

/*****************************************************************
 * REST_AddWi
 *
 * Description:
 *   This routine will add a work item to the global work item list.
 *
 * Parameters:
 *   wiName (I) - name of the work item to add
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_AddWi (Str wiName)
{
  RestoreWorkItemPtr newWI;  /* The new WI info */
  RestoreWorkItemPtr nextWI; /* Pointer to walk the current list with */

  /* Create the new work item */
  newWI = (RestoreWorkItemPtr) GUTIL_Malloc (sizeof(RestoreWorkItemRec));
  STR_Cpy (newWI->wiName, wiName);
  newWI->next = NULL;

  /* Attach the new work item to the end of the list */
  if (currentWIList != NULL)
  {
    nextWI = currentWIList;
    while (nextWI->next != NULL)
    {
      nextWI = nextWI->next;
    }
    nextWI->next = newWI;
  }
  else
  {
    currentWIList = newWI;
  }
}

/*****************************************************************
 * REST_UnmarkProgressCB
 *
 * Description:
 *   This routine will report current unmark progress to the user
 *
 * Parameters:
 *   totalMarks (I) - the number of unmarked items
 *
 * Returns:
 *   BOOL_TRUE  - If the unmark operation should continue
 *   BOOL_FALSE - If the user cancelled the operation
 *
 *****************************************************************/

static BoolEnum REST_UnmarkProgressCB (unsigned long totalMarks)
{
  Char      outputString[MAX_STRING_LENGTH];   /* Message string to display */
```



```
  STR_Sprintf (outputString,
               REST_GetErrorString (REST_UNMARK_PROGRESS_FORMAT),
               totalMarks);

  if (synchMarkHandle == NULL)
  {
    /* Initialize the window */
    synchMarkHandle = GALERT_DisplaySynchronousWait
                        ((WinPtr)REST_RestoreWin,
                         REST_GetErrorString (REST_UNMARK_PROGRESS_TITLE),
                         GICON_GetIcon(I_WAIT),
                         outputString,
                         BOOL_TRUE);

  }
  else
  {
    GALERT_UpdateMessage (synchMarkHandle, outputString);
  }

  /* Return whether or not the user cancelled */
  return (! GALERT_IsCancelled(synchMarkHandle));
}

/*****************************************************************
 * REST_MarkRestorableObject
 *
 * Description:
 *   This routine mark the given restore object.
 *
 * Parameters:
 *   restoreObject (I) - the object to mark
 *   backupTime    (I) - the time of the backup for the object
 *   numberMarked  (O) - the number of marked items
 *   numberBad     (O) - the number of bad items marked
 *
 * Returns:
 *   BOOL_TRUE  - If the object was marked successfully
 *   BOOL_FALSE - otherwise
 *
 *****************************************************************/

BoolEnum REST_MarkRestorableObject (GREST_Object  restoreObject,
                                    time_t        backupTime,
                                    long          *numberMarked,
                                    long          *numberBad)
{
  BoolEnum   marked = BOOL_FALSE; /* Flag if marking was successful */
  eerrno_ty  eerrno;              /* Error Status */
  Char       outputString[MAX_STRING_LENGTH];   /* Error string to display */
  u_long     badFiles = 0;        /* Number of bad files */
  u_long     permDeniedFiles = 0; /* Number of permission denied files */
  u_long     markedFiles = 0;     /* Number of marked files */
  u_long     markedDirs = 0;      /* Number of marked directories */
  u_long     markedOthers = 0;    /* Number of marked other types */
  u_long     totalMarks = 0;      /* Number of total marks */
  Str        fullName;            /* Full name of the object */
  boolean_ty interrupt = BOOL_FALSE; /* Flag to interrupt operation */

  /* Validate the object */
  if ((restoreObject != NULL) && (numberMarked != NULL) && (
                                                   numberBad != NULL))
  {

    /*
```

```
    * Check if an object by this name is alread selected and warn the
    * user before marking it.  This can happen if the user attempts to
    * mark the same file backed up at different times.
    */

  fullName = esl_strdup (EDMRST_GetObjectFullName (
                                                GREST_Handle, restoreObject));
  REST_StripDirectoryChars (fullName);

  if (REST_IsNameSelected (fullName))
  {
    /* Tell the user the object name is already marked */
    STR_Sprintf (outputString,
                 REST_GetErrorString (REST_NAME_MARKED_FORMAT),
                 fullName);

    /* Ask the user if he/she wants to continue anyways */
    if (GALERT_DisplayQuestion((WinPtr)REST_RestoreWin,
                               REST_GetErrorString (
                                          REST_NAME_MARKED_WARNING),
                               GICON_GetWarning(),
                               outputString,
                               BOOL_FALSE) != GALERT_Affirmative)
    {
      GUTIL_Free (fullName);
      return (BOOL_FALSE);
    }

  }

  /* Initialize the current mark handle to NULL */
  synchMarkHandle = NULL;

  /* If time zero was passed, mark at the current backup time */
  if (backupTime == 0)
  {

    /* Make sure the object is markable */
    if (!GREST_IsObjectMarkable (GREST_Handle, restoreObject))
    {

      /* tell the user it can't be marked */
      STR_Sprintf (outputString,
                   REST_GetErrorString (REST_MARK_ERROR),
                   fullName);
      /* Display the error message */
      GALERT_DisplayError ((WinPtr)REST_RestoreWin,
                           REST_GetErrorString (REST_ERROR_INDEX),
                           GICON_GetError(),
                           outputString);

      /* Return that the object wasn't marked */
      GUTIL_Free (fullName);
      return (BOOL_FALSE);
    }
  }

  /* Mark the object */
  eerrno = EDMRST_MarkObject (GREST_Handle,
                              restoreObject,
                              backupTime,
                              REST_MarkBadFiles,
                              BOOL_TRUE);

  if (eerrno == E_SUCCESS)
  {
```



```
    while ((eerrno = EDMRST_GetMarkResults (GREST_Handle,
                                            interrupt,
                                            &badFiles,
                                            &permDeniedFiles,
                                            &markedFiles,
                                            &markedDirs,
                                            &markedOthers)) ==
                                               EP_RB_RECOVER_RPC_INCOMPLETE)
    {
      totalMarks = markedFiles + markedDirs + markedOthers;
      if (totalMarks > REST_MARK_THRESHOLD)
      {
        STR_Sprintf (outputString,
                     REST_GetErrorString (REST_MARK_PROGRESS_FORMAT),
                     totalMarks);

        if (synchMarkHandle == NULL)
        {
          /* Initialize the window */
          synchMarkHandle = GALERT_DisplaySynchronousWait
                              ((WinPtr)REST_RestoreWin,
                               REST_GetErrorString (
                                          REST_MARK_PROGRESS_TITLE),
                               GICON_GetIcon(I_WAIT),
                               outputString,
                               BOOL_TRUE);

        }
        else
        {
          GALERT_UpdateMessage (synchMarkHandle, outputString);
        }

        /* Determine if the user cancelled yet */
        interrupt = GALERT_IsCancelled(synchMarkHandle);
      }
    }

    if (synchMarkHandle != NULL)
    {
      if (GALERT_IsCancelled (synchMarkHandle))
      {
        /* Display the warning message */
        GALERT_DisplayError ((WinPtr)REST_RestoreWin,
                             REST_GetErrorString (
                                          REST_MARK_CANCELLED_TITLE),
                             GICON_GetWarning(),
                             REST_GetErrorString (
                                          REST_MARK_CANCELLED_MESSAGE));
      }

      GALERT_CancelSynchDialog (synchMarkHandle);
      synchMarkHandle = NULL;
    }
  }

  /* If there is no error, update the marked data */
  if (eerrno == E_SUCCESS)
  {
    /* Set the marked and bad count */
    *numberMarked = markedFiles + markedDirs + markedOthers;
    if (REST_MarkBadFiles)
    {
      *numberBad = badFiles;
    }
    else
```

```
        {
          *numberBad = 0;
        }

        /* Successfully marked,  add the object to the selection list */
        REST_SelectRestorableItem (restoreObject, backupTime);
        marked = BOOL_TRUE;
      }

      /* Else display an error message to the user */
      else
      {
        STR_Sprintf (outputString,
                     REST_GetErrorString (REST_MARK_ERROR),
                     fullName);
        REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
                                  NULL,
                                  outputString,
                                  eerrno);
      }

      GUTIL_Free (fullName);
    }

    /* Return whether or not the object was marked */
    return (marked);
}

/*****************************************************************
 * REST_UnmarkRestorableObject
 *
 * Description:
 *   This routine unmark the given restore object.
 *
 * Parameters:
 *   restoreObject (I) - the object to mark
 *   backupTime    (I) - the time of the backup for the object
 *   numberMarked  (O) - the number of marked items
 *   numberBad     (O) - the number of bad items marked
 *
 * Returns:
 *   BOOL_TRUE  - If the object was unmarked successfully
 *   BOOL_FALSE - otherwise
 *
 *****************************************************************/

BoolEnum REST_UnmarkRestorableObject (GREST_Object  restoreObject,
                                      time_t        backupTime,
                                      long          *numberMarked,
                                      long          *numberBad)
{
  BoolEnum    unmarked = BOOL_FALSE; /* Flag if marking was successful */
  Char        outputString[MAX_STRING_LENGTH];  /* Error string to display */
  eerrno_ty   eerrno;                /* Error Status */
  u_long      badFiles = 0;          /* Number of bad files */
  u_long      markedFiles = 0;       /* Number of marked files */
  u_long      markedDirs = 0;        /* Number of marked directories */
  u_long      markedOthers = 0;      /* Number of marked other types */
  u_long      totalMarks = 0;        /* Number of total marks */
  boolean_ty interrupt = BOOL_FALSE; /* Flag to interrupt operation */

  /* Validate the object */
  if ((restoreObject != NULL) && (numberMarked != NULL) && (
                                                  numberBad != NULL))
  {
    /* Initialize the current mark handle to NULL */
```



```
    synchMarkHandle = NULL;

    eerrno = EDMRST_UnmarkObject (GREST_Handle,
                                  restoreObject,
                                  backupTime,
                                  BOOL_FALSE,
                                  BOOL_TRUE);
    if (eerrno == E_SUCCESS)
    {
      while ((eerrno = EDMRST_GetUnmarkResults (GREST_Handle,
                                                interrupt,
                                                &badFiles,
                                                &markedFiles,
                                                &markedDirs,
                                                &markedOthers)) ==
                                                 EP_RB_RECOVER_RPC_INCOMPLETE)
      {
        totalMarks = markedFiles + markedDirs + markedOthers;
        if (totalMarks > REST_MARK_THRESHOLD)

        {
          STR_Sprintf (outputString,
                       REST_GetErrorString (REST_UNMARK_PROGRESS_FORMAT),
                       totalMarks);

          if (synchMarkHandle == NULL)
          {
            /* Initialize the window */
            synchMarkHandle = GALERT_DisplaySynchronousWait
                              ((WinPtr)REST_RestoreWin,
                               REST_GetErrorString (
                                                REST_UNMARK_PROGRESS_TITLE),
                               GICON_GetIcon(I_WAIT),
                               outputString,
                               BOOL_TRUE);

          }
          else
          {
            GALERT_UpdateMessage (synchMarkHandle, outputString);
          }

          /* Determine if the user cancelled yet */
          interrupt = GALERT_IsCancelled(synchMarkHandle);
        }
      }

      if (synchMarkHandle != NULL)
      {
        if (GALERT_IsCancelled (synchMarkHandle))
        {
          /* Display the warning message */
          GALERT_DisplayError ((WinPtr)REST_RestoreWin,
                               REST_GetErrorString (
                                                REST_UNMARK_CANCELLED_TITLE),
                               GICON_GetWarning(),
                               REST_GetErrorString (
                                                REST_UNMARK_CANCELLED_MESSAGE));
        }

        GALERT_CancelSynchDialog (synchMarkHandle);
        synchMarkHandle = NULL;
      }
    }

    /* If there is no error, update the marked data */
```

```
      if (eerrno == E_SUCCESS)
      {
        unmarked = BOOL_TRUE;
        *numberMarked = markedFiles + markedDirs + markedOthers;
        if (REST_MarkBadFiles)
        {
          *numberBad = -badFiles;
        }
        else
        {
          *numberBad = 0;
        }

        /* Deselect the object */
        REST_DeselectInfo (restoreObject, backupTime);
      }

      /* Else display an error message to the user */
      else
      {
        Char        outputString[MAX_STRING_LENGTH];  /* Error output string */

        /* Create the error string */
        STR_Sprintf (outputString,
                     REST_GetErrorString (REST_UNABLE_TO_UNMARK),
                     EDMRST_GetObjectFullName (GREST_Handle, restoreObject));

        /* Display the error message */
        REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
                                  NULL,
                                  outputString,
                                  eerrno);
      }
    }

    /* Return whether or not the object was unmarked */
    return (unmarked);
}

/*****************************************************************
 * REST_MarkInfo
 *
 * Description:
 *   This routine will unmark the given info object
 *
 * Parameters:
 *   info         - The info object to unmark
 *   numberMarked - The number of marked objects
 *   numberBad    - The number of bad files unmarked
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

BoolEnum REST_MarkInfo (RestoreInfoPtr  info,
                        long            *numberMarked,
                        long            *numberBad)
{
  BoolEnum          thisMark = BOOL_FALSE;
                                /* Flag if this mark was successful */
  long              thisBad = 0;          /* Number of bad files in a mark */
  RestoreInfoPtr    nextChild;            /* Loop pointer to next object */

  if (info != NULL)
  {
```



```
    /* If this is a file or a directory mark the restorable object */
    if ((!info->marked) &&
        ((info->type == REST_File) || (info->type == REST_Directory)))
    {
      thisMark = REST_MarkRestorableObject (info->restoreObject,
                                            0,
                                            numberMarked,
                                            numberBad);
    }
    else if ((!info->marked) && (info->type == REST_WorkItem))
    {
      if (info->children == NULL)
      {
        /* Set the flag so that the objects aren't displayed */
        updatingDate = BOOL_TRUE;

        /* Add the child objects */
        REST_CreateInfoChildren (info);

        /* Set the flag back */
        updatingDate = BOOL_FALSE;
      }

      /* Loop through and mark all the children */
      nextChild = info->children;
      while (nextChild != NULL)
      {

        /* Mark this child */
        if (!nextChild->marked)
        {
          thisMark |= REST_MarkRestorableObject (nextChild->restoreObject,
                                                 0,
                                                 numberMarked,
                                                 &thisBad);

          /* Add in the number of bad files for this mark */
          *numberBad = *numberBad + thisBad;
        }

        /* Move on to the next child */
        nextChild = nextChild->next;
      }
    }

    /* Update the mark flags for all objects */
    REST_UpdateObjectMarks (currentWorkItemInfo);
  }

  return (thisMark);
}

/*****************************************************************
 * REST_UnmarkInfo
 *
 * Description:
 *   This routine will unmark the given info object
 *
 * Parameters:
 *   info         - The info object to unmark
 *   numberMarked - The number of marked objects
 *   numberBad    - The number of bad files unmarked
 *
 * Returns:
 *   None.
 *
 *****************************************************************/
```

```
BoolEnum REST_UnmarkInfo (RestoreInfoPtr  info,
                          long            *numberMarked,
                          long            *numberBad)
{
  BoolEnum        thisMark = BOOL_FALSE;
                                          /* Flag if this mark was successful */
  eerrno_ty       eerrno;                  /* Error status */
  time_t          currentBackupTime;       /* Time of the current backup */
  long            thisBad = 0;            /* Number of bad files in a mark */
  RestoreInfoPtr nextChild;               /* Loop pointer to next object */

  if (info != NULL)
  {
    /* remove the item from the list */
    if ((info->marked) &&
        ((info->type == REST_File) || (info->type == REST_Directory)))
    {
      /* Set the backup time to the current backup time */
      if ((eerrno = EDMRST_GetCurrentBackupTime(
                                    GREST_Handle, &currentBackupTime)) != 0)
      {
        /* Hmm... I guess we just use time zero */
        currentBackupTime = 0;
      }

      thisMark = REST_UnmarkRestorableObject (info->restoreObject,
                                              currentBackupTime,
                                              numberMarked,
                                              numberBad);

    }
    else if ((info->marked) && (info->type == REST_WorkItem))
    {
      /* Set the backup time to the current backup time */
      if ((eerrno = EDMRST_GetCurrentBackupTime(
                                    GREST_Handle, &currentBackupTime)) != 0)
      {
        /* Hmm... I guess we just use time zero */
        currentBackupTime = 0;
      }

      if (info->children == NULL)
      {
        /* Set the flag so that the objects aren't displayed */
        updatingDate = BOOL_TRUE;

        /* Add the child objects */
        REST_CreateInfoChildren (info);

        /* Set the flag back */
        updatingDate = BOOL_FALSE;
      }

      /* Loop through and mark all the children */
      nextChild = info->children;
      while (nextChild != NULL)
      {
        /* Mark this child */
        thisMark = REST_UnmarkRestorableObject (nextChild->restoreObject,
                                                currentBackupTime,
                                                numberMarked,
                                                &thisBad);
        /* Add in the number of bad files for this mark */
        *numberBad = *numberBad + thisBad;

        /* Move on to the next child */
```



```
        nextChild = nextChild->next;
      }
    }

    /* Update the mark flags for all objects */
    REST_UpdateObjectMarks (currentWorkItemInfo);
  }

  return (thisMark);
}

/*****************************************************************
 * REST_Initialize
 *
 * Description:
 *   This routine will initialize all componets of restore. It will
 *   load the window resources and set up all default values.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_Initialize (void)
{
  REST_SortType  sortType;
                                                  /* Initial sort type */
  Str            hostName;                         /* Client string */
  Char           windowLabel[2 * GMAX_HOSTNAME_LENGTH];
                                                  /* New window label */

  /* Start off clean */
  currentWorkItemInfo = NULL;

  /* Initialize the option flags */
  REST_ShowHiddenFiles = BOOL_TRUE;
  REST_ShowBadFiles = BOOL_FALSE;
  REST_MarkBadFiles = BOOL_FALSE;

  /* Initialize the File Manager */
  GFMGR_Initialize ();

  /* Load the restore window data */
  REST_RestoreWinLoadInit ();

  /* Get the string list of trail names */
  REST_TrailNameList  = (StrLPtr)RES_LoadInit ("restore", "TrailNames");

  /* Set up the tabs */
  REST_TabObject = GTAB_objInit (REST_RestoreWin->TabsPanel);
  GTAB_addPanelPair(REST_TabObject,
                    (TButPtr)REST_RestoreWin->MarkSummaryTab,
                    REST_RestoreWin->MarkSummaryPanel);
  GTAB_addPanelPair(REST_TabObject,
                    (TButPtr)REST_RestoreWin->MediaTab,
                    REST_RestoreWin->MediaPanel);
  GTAB_addPanelPair(REST_TabObject,
                    (TButPtr)REST_RestoreWin->ViewOptionsTab,
                    REST_RestoreWin->ViewOptionsPanel);

  /* Get the icons used */
  REST_ClientIcon      = GICON_GetIconBySize (I_GENERICCLIENT, ICON_SMALL);
  REST_FSWorkItemIcon = GICON_GetIconBySize (I_FILESYSWI, ICON_SMALL);
```

```
    REST_FailedWorkItemIcon = GICON_GetIconBySize (
                                                    I_UNUSED_WORKITEM, ICON_SMALL);
    REST_DirClosedICon      = GICON_GetIconBySize (I_DIRCLOSED, ICON_SMALL);
    REST_DirOpenIcon        = GICON_GetIconBySize (I_DIROPEN, ICON_SMALL);
    REST_FileIcon           = GICON_GetIconBySize (I_FILE, ICON_SMALL);
    REST_CheckIcon          = GICON_GetIconBySize (I_CHECK, ICON_SMALL);
    REST_BadIcon            = GICON_GetIcon (I_BADBUOBJECT);
    REST_ExpiredIcon        = GICON_GetIconBySize (I_ERROR, ICON_SMALL);
    REST_MissingChildrenIcon = GICON_GetIconBySize (I_WARNING, ICON_SMALL);
    REST_FailedIcon         = GICON_GetIconBySize (I_ERROR, ICON_SMALL);

    /* Initialize the other restore pieces */
    RESTutilInitialize ();
    REST_FileInitialize ();
    REST_SearchInitialize ();
    REST_SelInitialize ();

    /* Initialize the sort radio buttons to the initial sort type */
    sortType = REST_GetSort ();
    switch (sortType)
    {
      case REST_ByName:
        TBUT_SetSelected ((TButPtr)REST_RestoreWin->NameSortButton, BOOL_TRUE);
        break;
      case REST_ByType:
        TBUT_SetSelected ((TButPtr)REST_RestoreWin->TypeSortButton, BOOL_TRUE);
        break;
      case REST_ByDate:
        TBUT_SetSelected ((TButPtr)REST_RestoreWin->DateSortButton, BOOL_TRUE);
        break;
      case REST_BySize:
        TBUT_SetSelected ((TButPtr)REST_RestoreWin->SizeSortButton, BOOL_TRUE);
        break;
      case REST_ByOwner:
        TBUT_SetSelected ((TButPtr)REST_RestoreWin->OwnerSortButton, BOOL_TRUE);
        break;
      default:
        TBUT_SetSelected ((TButPtr)REST_RestoreWin->NameSortButton, BOOL_TRUE);
        break;
    }

    /* Initialize the options to their initial states */
    TBUT_SetSelected ((TButPtr)REST_RestoreWin->HiddenButton,
                      REST_ShowHiddenFiles);
    TBUT_SetSelected ((TButPtr)REST_RestoreWin->BadFilesButton,
                      REST_ShowBadFiles);
    TBUT_SetSelected ((TButPtr)REST_RestoreWin->MarkBadButton,
                      REST_MarkBadFiles);

    /* Determine if we want to show the client name */
    if (REST_RestoreFromClient)
      hostName = REST_RestoreClient;
    else
      hostName = NULL;

    /* Set the Window and Icon Labels */
    GUTIL_SetDefaultWindowTitle ((WinPtr)REST_RestoreWin, hostName);
    WIN_SetIcon ((WinPtr)REST_RestoreWin,
                 GICON_GetIconBySize (I_RESTORE, ICON_LARGE));

    GUTIL_CBOX_AllowFreeStyle (
                            REST_RestoreWin->TemplateBox,BACKUP_EXCLUDE_CHARS);
}

/*****************************************************************
 * REST_Display
```



```
 *
 * Description:
 *   This routine will display the restore window in an initial state.
 *   This can be used the first time the window is displayed or whenever
 *   the window needs to be reset to the initial state.  It assumes the
 *   window resources have already been loaded.  It does not reset user
 *   selections for preferences.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_Display (void)
{
  eerrno_ty         eerrno;     /* Error Status */

  /* Reset the current info */
  currentWorkItemInfo = NULL;

  /* Initialize this restoral */
  if ((eerrno = EMXRST_Initialize (GUTIL_GetThisHost (),
                                   &GREST_Handle,
                                   CONNECT_TIMEOUT_SEC)) != E_SUCCESS)
  {

    /* Couldn't initialize, nothing we can do! */
    REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
                              NULL,
                              REST_GetErrorString(REST_INIT_FAILURE),
                              eerrno);

    _exit (-1);
  }

  /* Set buttons and labels to default values */
  TBUT_SetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton, BOOL_TRUE);
  TED_SetStr ((TEdPtr)REST_RestoreWin->RestoreItemsTarea, "");
  TED_SetStr ((TEdPtr)REST_RestoreWin->RestoreSizeTarea, "");
  TED_SetStr ((TEdPtr)REST_RestoreWin->BadFilesText, "");

  /* Start off with any clients */
  REST_ShowClients (&currentClientList, currentWIList);

  /* Initialize the state of the buttons */
  REST_LBoxSelectionNfy ( NULL );
  REST_UpdateRemoveButtons ();

  /* Nothing is selcted */
  REST_SetSelectionString ("");

  /* Display the file manager */
  GFMGR_Display (REST_GetFMgrContext());

}

/*****************************************************************
 * REST_ClearSession
 *
 * Description:
 *   This routine will free up all memory and clear all lists assosicated
 *   with the current restore session.  If the resetFlag is true, then the
 *   current marks will be unmarked. If False, we are exitting so there is
 *   no need.
```

```
  *
  * Parameters:
  *   resetFlag - Flag if the session is being reset (versus exit)
  *
  * Returns:
  *   None.
  *
  ************************************************************************/

 void REST_ClearSession (BoolEnum  resetFlag)
 {
   RestoreInfoPtr  thisInfo;  /* Pointer to walk the list with */
   RestoreInfoPtr  nextInfo;  /* Next Pointer in the list */

   /* If this is a reset operation, clear the marks and media */
   if (resetFlag)
   {
     /* Clear out the list boxes */
     REST_RemoveAllSelectedItems ();
     REST_RemoveAllMedia ();
   }

   /* Clear out the file manager */
   GFMGR_ClearAll (REST_GetFMgrContext());

   /* Free up each top level object (will recursively free up all children) */
   thisInfo = (RestoreInfoPtr) GFMGR_GetVeryFirstObject (REST_GetFMgrContext());
   while (thisInfo != NULL)
   {
     nextInfo = thisInfo->next;
     REST_FreeInfo (thisInfo);
     thisInfo = nextInfo;
   }

   /* Finalize the Restore process */
   EDMRST_Finish (GREST_Handle);
   GREST_Handle = NULL;

   /* Remove the search window if it is up */
   REST_SearchRemove ();

 }

 /************************************************************************
  * REST_Remove
  *
  * Description:
  *   This routine will remove the restore dialog from the display after
  *   verification with the user.
  *
  * Parameters:
  *   None.
  *
  * Returns:
  *   BOOL_TRUE  - If the restore window was really removed
  *   BOOL_False - Otherwise
  *
  ************************************************************************/

 BoolEnum REST_Remove (void)
 {
   WinPtr    parent;
   BoolEnum  OKToExit;  /* Flag if user really wants to exit */

   /* Don't allow exit if we're searching, or a restore is in progress */
   if (REST_SearchInProgress() || REST_RestoreInProgress())
```



```
     return (BOOL_FALSE);

   /* Use the restore window if it is currently visible */
   if (WIN_IsOpen ((WinPtr)REST_RestoreWin))
     parent = (WinPtr)REST_RestoreWin;
   else
     parent = NULL;

   /* Verify that the user really wants to end the session */
   OKToExit =  (GALERT_DisplayQuestion(parent,
                                       REST_GetErrorString(REST_WARNING_INDEX),
                                       GICON_GetQuestion(),
                                       REST_GetErrorString (REST_IS_OK_TO_END),
                                       BOOL_FALSE) == GALERT_Affirmative);

   /* If the user says so, go ahead with the termination */
   if (OKToExit)
   {
     REST_ClearSession (BOOL_FALSE);
   }

   return (OKToExit);
 }

 void REST_SignalHandler (int sig)
 {
     /* Clean up help */
     EDMHELP_End();

     /* Call the generic signal handler to clean up */
     GUTIL_GenericSignalHandler(sig);
 }
```

```
/*****************************************************************************
**
** File Name:   RSTinitfin.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**              This module contains the Restore API functions to
**              initialize and terminate the restore operation.
**
** Table of Contents:
** ------------------
**      API Functions:
**              EDMRST_Initialize
**              EDMRST_Finish
**
**      Internal Functions:
**
**
** Compile-Time Options:
**              This section must list any compile time definitions
**              which will affect this header.
**
*****************************************************************************/


/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1

/*
 * System headers.
 */
#include <pwd.h>


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>

/*
 * Local headers
 */
#include <RSTinterns.h>
#include <RSTsup_csm.h>
```



```
/*
 * Comms headers.
 */
#include <restore/csc_EDMDispatch.h>
#include <restore/csc_EDMRestoreEng.h>
#include <restore/dispatch_daemon.h>
#include <restore/restore_engine.h>
#include <edmlink/edmlink_api.h>

/*
 * #defines, structures, typedefs local to this source file
 */


/*
 * Global declarations
 */

    internalHandlePtr handlePtr = NULL;
```

```
/*********************************************************************
 * EDMRST_Initialize:
 *
 *  This function takes care of all the initialization for a recovery
 *  session.  This must be called prior to any of the other functions
 *  in the Recover API.
 *
 * Parameters:
 *
 *  hostname    (I) - The machine name of the server to use.
 *  svrHdl      (O) - A handle to receive a pointer to this user's client
 *                    handle for the Restore Engine connection.
 *  timeout     (I) - The maximum number of seconds to wait for the connection
 *                    to the Restore Engine process to be completed.
 *
 *********************************************************************/

eerrno_ty
EDMRST_Initialize( hostname_ty   hostname,
                   serverHandle  *svrHdl,
                   unsigned long timeout )
{
        eerrno_ty api_status = E_SUCCESS;

        uid_t   human_uid;
        struct  passwd *pw;
        char    *human_uidname ;

        RE_initialize_args      re_init_args;
        RE_status_result        *re_init_result;
        rpc_if_handle_t                 re_if_spec;
        rpc_binding_handle_t    re_handle;
        int                     retval;
        time_t                  end_time;

#ifdef DEBUG
#define  RPC_TIMEOUT     3600
        struct timeval  rpc_timeout;
#endif

/********************** BEGINNING OF Dispatch Daemon STUFF ****************/
    error_status_t status;
    DD_initialize_args    initargs;
    DD_getservicestatus_args  statargs;
    DD_initialize_result  *initres = NULL;
    DD_getservicestatus_result  *statres = NULL;
    rpc_if_handle_t       if_spec;


    time( &end_time );   /* compute time to give up waiting */
    end_time += timeout;

    memset(&if_spec,0,sizeof(rpc_if_handle_t));
    memset(&re_if_spec,0,sizeof(rpc_if_handle_t));

    if (svrHdl == NULL || hostname == NULL )
    {
        return( EP_RB_RECOVER_BAD_ARGS );
    }

         rec_api_log_begin( "edmrestore_api" );
                                                /* init logs, ignore errs?? */

         /* get user name to pass to DD and RE */
         human_uid = getuid();
         pw = getpwuid( human_uid );
```



```
        if (pw == NULL || NULL == pw->pw_name )
        {
                /* Trouble.  */

                rec_api_log_csm(SUB_CSM_USER_NOT_IN_PASSWD, NULL);
                return(EP_RB_RECOVER_PERMISSION_DENIED);
        }

        human_uidname = pw->pw_name;

    handlePtr = (internalHandle *) calloc(1, sizeof(internalHandle));

    /* Use this macro to setup the interface spec */
    CLIENT_IFSPEC(if_spec);

    /* Arrive at a server binding.  Note that if they didn't give us
    ** a valid host parameter, this will fail and drop through and
    ** return NULL in the end.
    ** This call will get and store a fully resolved binding
    ** handle to the host.  The first time we ever call the host,
    ** csc_get_handle will resolve and store the binding.  If we
    ** ever use csc_get_handle to talk to the same host again,
    ** it will just give back the previously resolved binding.
    */

    retval = csc_get_handle((unsigned char *) hostname,
                            if_spec,
                            SERVER_GROUP,
                            &handlePtr -> dd_binding_handle,
                            &status);

    /*
    ** Find out if we got csc handle and see if status is bad.
    ** error_status_ok is a macro defined in cscomm.h.
    */

    if ((status != error_status_ok) || (retval == 0))
    {
      /*  If errno not set, use status if it is a valid errno value */

      if ( errno == 0 )
        errno = ( strerror( status ) ? status : ETIME );

      rec_api_log_csm( SUB_CSM_RPC_FAIL,
                       "failure finding edmdispd to start restore engine" );

      return EP_RB_RECOVER_SERVERFAIL;
    }

    errno = 0;

#ifdef DEBUG
/* increase rpc timeout during debugging */
    rpc_timeout.tv_sec = RPC_TIMEOUT;
    rpc_timeout.tv_usec = 0;
    clnt_control( handlePtr->dd_binding_handle, CLSET_TIMEOUT,
                  (char *)&rpc_timeout );
#endif

    initargs.service = DD_SERVICE_RESTORE;
    initargs.hostname = hostname;
    initargs.username = human_uidname;
    initargs.timeout = timeout;

    initres = dd_initialize_1( &initargs, handlePtr -> dd_binding_handle );
                /* Will have _1 for RPC call  */
```

```
    if (initres == NULL)
    {
        return EP_RB_RECOVER_RPC_FAIL;
    }

    statargs.service_handle = initres -> service_handle;
    statargs.status = 0;

    statres = dd_getservicestatus_1( &statargs, handlePtr->dd_binding_handle );

    if (statres == NULL)
    {
        return EP_RB_RECOVER_RPC_FAIL;
    }

    while (statres -> status == DD_SERVICE_STARTING )
    {
        time_t  now;

        xdr_free( xdr_DD_getservicestatus_result, (char *)statres );
        time( &now );
        if (now >= end_time)
        {
                rec_api_log_csm( SUB_CSM_RPC_FAIL,
                        "timeout waiting for edmdispd to start restore engine"
                                                                        );
                return EP_RB_RECOVER_SERVERFAIL;
        }

       sleep(1);

       statres = dd_getservicestatus_1( &statargs,
                                        handlePtr -> dd_binding_handle );

       if (statres == NULL)
       {
            rec_api_log_csm( SUB_CSM_RPC_FAIL,
          "failure getting status from edmdispd while starting restore engine" );
           return EP_RB_RECOVER_RPC_FAIL;
       }
    }

    if (statres -> status != DD_SERVICE_RUNNING)
    {
        rec_api_log_csm( SUB_CSM_RPC_FAIL,
                         "edmdispd failure while starting restore engine" );
        xdr_free( xdr_DD_getservicestatus_result, (char *)statres );
       return EP_RB_RECOVER_SERVERFAIL;
    }

    memcpy( handlePtr -> opaque128,
            statres -> handle.handle_val,
            sizeof(handlePtr -> opaque128) );

    xdr_free( xdr_DD_getservicestatus_result, (char *)statres );

/********************** END OF Dispatch Daemon STUFF ***************/

/* Restore Engine FUNCTIONALITY BEGINS HERE */

/*      RE_CLIENT_IFSPEC(re_if_spec);  */

       retval = csc_private_ifspec_init(
                                (unsigned char *) handlePtr -> opaque128,
                                RE_PROGNUM,
```



```
                                RE_VERSNUM,
                                &re_if_spec,
                                &status);

    if (retval == 0)
    {
        rec_api_log_csm( SUB_CSM_RPC_FAIL,
                    "failure initializing interface to restore engine"
                                                                        );
        return EP_RB_RECOVER_SERVERFAIL;
    }

    api_status = EP_RB_RECOVER_SERVERFAIL;
    do {
            time_t  now;
            time( &now );
            if (now >= end_time)
            {
                rec_api_log_csm( SUB_CSM_RPC_FAIL,
                        "timeout connecting to restore engine" );
                return EP_RB_RECOVER_SERVERFAIL;
            }

            sleep( 1 );     /* give restore engine time to get going */
            retval = csc_connect_to_rpc_service(
                                (unsigned char *)hostname,
                                re_if_spec,
                                RE_CLIENT_GROUP,
                                &handlePtr -> re_binding_handle,
                                &status);

            if ((status == error_status_ok) && (retval != 0))
                    api_status = E_SUCCESS;
    } while (api_status != E_SUCCESS);

    if (api_status == E_SUCCESS)
    {
            re_handle = handlePtr -> re_binding_handle;
#ifdef DEBUG
/*      increase rpc timeout during debugging */
            rpc_timeout.tv_sec = RPC_TIMEOUT;
            rpc_timeout.tv_usec = 0;
            clnt_control( re_handle, CLSET_TIMEOUT, (
                                                char *)&rpc_timeout );
#endif
            re_init_args.username = human_uidname;
            set_rpc_obj( re_initialize, &re_init_args.RPCobjID );
            re_init_result = re_initialize_1( &re_init_args, re_handle );
            if (!re_init_result) {
                    api_status = EP_RB_RECOVER_RPC_FAIL;
                    rec_api_log_csm( SUB_CSM_RPC_FAIL,
                            "failure communicating with restore engine" );
            }
            else {
                    api_status = re_init_result->status;
                    /* release RPC result struct: */
                    xdr_free( xdr_RE_status_result, (
                                                char *)re_init_result);
            }
    }

    else
        rec_api_log_csm( SUB_CSM_RPC_FAIL,
                        "failure connecting to restore engine" );

    if (
      api_status == E_SUCCESS)     /* return rest eng handle on success */
```

```
                *svrHdl = (serverHandle)re_handle;

        return( api_status );

}       /* End of EDMRST_Initialize() */
```



```
/*****************************************************************
 * Ping:
 *
 *  This function allows a ping to be issued in order to keep the
 *  engine alive and running so that the engine will not time out.
 *
 * Parameters:
 *
 *  svrHdl (I) - A pointer to this user's client handle for the
 *               Restore Engine (server) connection.
 *
 *****************************************************************/
eerrno_ty EDMRST_Ping( serverHandle svrHdl )
{
  eerrno_ty      api_status = E_SUCCESS;
  RE_null_args   re_ping_args;
  RE_status_result       *re_ping_result = NULL;


  if ( NULL == svrHdl || NULL == handlePtr
       || svrHdl != handlePtr->re_binding_handle )
    {
      return( EP_RB_RECOVER_BAD_ARGS );
    }

  set_rpc_obj( re_ping, &re_ping_args.RPCobjID );
  re_ping_result = re_ping_1( &re_ping_args, svrHdl );
  if (NULL == re_ping_result) {
    api_status = EP_RB_RECOVER_RPC_FAIL;
    rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
  }
  else {
    api_status = re_ping_result->status;
    /* release RPC result struct: */
    xdr_free( xdr_RE_status_result, (char *)re_ping_result);
  }

}



/*****************************************************************
 * EDMRST_Finish
 *
 *    Function Description:
 *
 *  This function terminates a restoral session,
 *                                        but only during the browse and
 *  mark phase. It will be rejected if a restore is currently being executed.
 *  This routine will clean up any local memory used in the session and will
 *  disconnect from the Restore Engine.  After calling this function,
 *  EDMRST_Initialize MUST be called before calling any other functions in
 *                                                                    this
 *  API.
 *
 * Parameters:
 *
 *  svrHdl (I) - A pointer to this user's client handle for the
 *               Restore Engine (server) connection.
 *
 *    Return Codes:
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_RPC_FAIL
 *       EP_RB_RECOVER_INVALOP
 *       EP_RB_RECOVER_SERVERFAIL
 *
```

```
 */

eerrno_ty
EDMRST_Finish( serverHandle svrHdl )
{
        eerrno_ty       api_status = E_SUCCESS;
        RE_null_args    re_finish_args;
        RE_status_result        *re_finish_result = NULL;
        int             csc_status;

        if ( NULL == svrHdl || NULL == handlePtr
          || svrHdl != handlePtr->re_binding_handle )
        {
                return( EP_RB_RECOVER_BAD_ARGS );
        }

        set_rpc_obj( re_finish, &re_finish_args.RPCobjID );
        re_finish_result = re_finish_1( &re_finish_args, svrHdl );
        if (!re_finish_result) {
                api_status = EP_RB_RECOVER_RPC_FAIL;
                rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
        }
        else {
                api_status = re_finish_result->status;
                /* release RPC result struct: */
                xdr_free( xdr_RE_status_result, (char *)re_finish_result);
        }

        rec_api_log_end();      /* write last log and close the log file. */


        return( api_status );

}       /* EDMRST_Finish */
```

```
 /*
 ** Copyright 1996,1997 EMC Corporation
 */

 /*
 ** EDMmain.c
 **
 ** Mission Statement: This is the main service file for the EDMsession daemon.
 **                    This file contains the main loop,
                                                    and all calls required
 **                    to prepare the daemon to go off and service RPC's.
 **
 ** Primary Data Acted On:
 **
 ** Compile-Time Options:
 **
 **                    USE_SUNRPC - Compile source with sunrpc support.  If
 **                                 not set, assume DCE support.
 **
 **                    NONPRODUCTION - Compile source for in house,
                                                          developer
 **                                    testing on local work station.
                                                            Should
 **                                    only be used for targted testing.
 **
 ** Basic idea here: Initalize required locks, establish signal handlers,
 **                  register RPC interface, go wait for RPCs.
 */

/*
** The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/
#if !defined(lint)
static char     RCS_id [] = "@(#)$RCSfile$ "
                            "$Revision$ "
                            "$Date$" ;
#endif

/* #define _POSIX_SOURCE   unable to compile with this define set */
/* #define _XOPEN_SOURCE   unable to compile with this define set */

/*****************************************************************************
**
** Routine: main
**
** Inputs: argc, argv
**
** Outputs:  None
**
** Return Codes:
**              exit status
**
** Purpose:     This is the main routine which sets up the daemon
**               to handle RPC calls, and handles them until it is told
**               to stop or it sees a fatal error.
**
** Intended caller:  None
**
*****************************************************************************
*/
main (int argc, char *argv[])
{
  /*
  ** Parse options
  */
```



```
  (void) parse_commandline(argc, argv);

  /*
  ** Setup logging
  */

  (void) daemon_initialize_logging();

  /*
  ** Enable permanent interrupt catching
  */

  (void) daemon_catch_interrupts();

  /*
  ** Function may not return if improper user running daemon
  */

  (void) daemon_check_proper_ID();

  /*
  ** Function will not return if this fails
  */

  (void) daemon_become_daemon();

  /*
  ** Re-establish log initialization since all *fd's* were
  ** closed by esl_daemon_startup (in daemon_become_daemon)
  */

  (void) daemon_initialize_logging();

  /*
  ** This function doesn't return on failure
  */

  (void) daemon_specific_initialization();

  /*
  ** Unregister service, cleanup cache... Never returns...
  */

  (void) daemon_cleanup();

  /*
  **  Strictly to inhibit compiler warning...
  */

  return( 0 );
}
```

```
/*
** Copyright 1996,1997 EMC Corporation
*/

/*
** EDMRestoreEng.c
**
** Mission Statement: This is the main service file for the EDMsessd daemon.
                                                                    This
**                    file contains the callbacks from the main function which
**                    prepares the daemon to go off and service RPC's.
**
** Primary Data Acted On:
**
** Compile-Time Options:
**
**                    USE_SUNRPC - Compile source with sunrpc support.  If
**                                 not set, assume DCE support.
**
** Basic idea here: Module for UNIX specific daemon initialization
*/

/*
** The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/
#if !defined(lint)
static char      RCS_id [] = "@(#)$RCSfile: EDMsessdd.c,v $ "
                             "$Revision: 1.23 $ "
                             "$Date: 1997/02/06 20:49:15 $" ;
#endif

/* #define _POSIX_SOURCE    unable to compile with this define set */
/* #define _XOPEN_SOURCE    unable to compile with this define set */

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <esl/inout.h>

#include <stdarg.h>
#include <string.h>
#include <syslog.h>
#include <pthread.h>
#include <thread.h>
#include <sys/utsname.h>
#include <netdb.h>

#include <logging/logging.h>
#include <util/esl_core.h>
#include <util/esl_pidfile.h>
#include <util/esl_daemon.h>
#include <csc/cscomm.h>

#include <restore/csc_EDMRestoreEng.h>

#include <EDMmain.h>
#include <EDMRestoreEngLog.h>
#include <EDMProcessManager.h>
#include <EDMProgress.h>
#include <EDMRE_ccr.h>
#include <EDMRE_ccw.h>
#include <EDMRECommandApi.h>
#include <EDMREQuestionApi.h>
#include <EDMREDrainApi.h>

/*
```



```
 * Need to define _XOPEN_SOURCE for signal funtion definitions
 * and certain signal structure definitions.
 */
#define _XOPEN_SOURCE

#include <signal.h>

#undef _XOPEN_SOURCE

static rpc_if_handle_t if_spec;

static int       G_debug = FALSE; /* Variable which will disable forking */

static char      **commandlineargs; /* Pointer to command line args */

/*****************************************************************************
**
** Routine: IsDebugOn
**
** Inputs: None
**
** Outputs:  None
**
** Return Codes:
**              TRUE if debug is on.
**
** Purpose:     This routine can be used to tell other subsystems
**              whether debugging is available.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/
boolean_ty
IsDebugOn()
{
#ifdef DEBUG
    return TRUE;          /* if DEBUG defined, we must be in debug mode */
#endif
    if (debugmode)        /* if turned on manually via adb, its on */
        return TRUE;

    return G_debug;       /* default is how we were started: -d menas debug */
}

/*****************************************************************************
**
** Routine:kill_handler
**
** Inputs: int sigval - the signal which was receieved.
**
** Outputs:  Will log messages telling what action is being taken.
**
** Return Codes:
**              exits with the number of the signal received
**
** Purpose:     This routine handles specific signals i.e. SIGINT,
**              SIGQUIT, SIGTERM. Each results in a log entry and an exit.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/
static void kill_handler( IN int sigval )
{
error_status_t   status;
```

```
time_t          current_time;
char            *ctimebuf;
char            *ebuff = NULL;

  /* If main exits, it calls this routine with signal 0 */

  /* Unregister the interface */
  (void) csc_unregister_server_interface(&if_spec, &status);

  /* If the unregister fails, report the problem, but continue */
  if ( status != error_status_ok )
  {
    ebuff = (char *) csc_get_error( status );

    (void) EDMRestoreEng_logent(
                        __FILE__,__LINE__, LOG_ERR, MESSAGE_NO_LOGIN, 0,
                        "CSC_SERVER_LOGIN failed: <%d> %s",
                        status, (ebuff ? ebuff : "Unknown error") );
  }

  /* Get the current time */
  (void) time(&current_time);

  ctimebuf = ctime(&current_time);

  /* Overlay newline with null - buf should always be 26 bytes long */
  ctimebuf[ strlen(ctimebuf) - 1 ] = 0;

  (void) EDMRestoreEng_logent(
                        __FILE__,__LINE__, LOG_INFO, MESSAGE_SHUTDOWN, 0,
                    "Shutting down at %s due to signal %d", ctimebuf, sigval);

  exit(sigval);

} /* End of kill_handler() */


/*****************************************************************************
**
** Routine: unregister_csc
**
** Inputs: none
**
** Outputs:  Will log messages telling what action is being taken.
**
** Return Codes:
**              none
**
** Purpose:     This routine handles the csc_unregister call
**
** Intended caller:  internal and process manager before exit
**
*****************************************************************************
*/
void unregister_csc( void )
{
error_status_t   status;
char     *ebuff = NULL;

  /* Unregister the interface */
  (void) csc_unregister_server_interface(&if_spec, &status);

  /* If the unregister fails, report the problem, but continue */
  if ( status != error_status_ok )
  {
    ebuff = (char *) csc_get_error( status );
```



```
    (void) EDMRestoreEng_logent(        __FILE__,__LINE__, LOG_ERR,
                        MESSAGE_CANNOT_UNREGISTER, 0,
                        "CSC_UNREGISTER_SERVER failed: <%d> %s",
                        status, (ebuff ? ebuff : "Unknown error") );
  }

  return;
}

/*****************************************************************************
 * Function Name:
 *       display_usage
 *
 * Simply displays the usage
 *
 * Call Arguments:
 *       Program name
 *
 * Error Outputs and Side Effects:
 *       Prints usage.
 *
 * Special Considerations:
 *       None.
 *
 *****************************************************************************
 */
static void
display_usage (IN char *progname)
{
  /* Print out usage stmt.  */

  fprintf (stderr,"Usage: %s [-d]\n", progname);
  fprintf (stderr,"-d keep the daemon from forking so debugging is easier\n");

} /* end display_usage () */

/*****************************************************************************
**
** Routine: daemon_catch_interrupts
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     Sets up signals for service. On NT we will have to
**              consider what OS constructs to replace signals with.
**              In this case we are catching SIGTERM, SIGINT, and
**              SIGQUIT and ignoring anything else.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/
void daemon_catch_interrupts()
{
struct sigaction          sactions;       /* Signal actions */

        ZERO( sactions );

        /*
         * Set an empty list so we can set signals we want to handle
```

```
        */
        (void) sigemptyset( &sactions.sa_mask );

       /*
        * Add signals that we want to handle
        */
        (void) sigaddset( &sactions.sa_mask, SIGTERM );
        (void) sigaddset( &sactions.sa_mask, SIGINT );
        (void) sigaddset( &sactions.sa_mask, SIGQUIT );

        /* Setup the signal handler. */
        sactions.sa_handler = kill_handler;

       /*
        * Assign handler to each signal we are interested in.
        */
        (void) sigaction( SIGTERM, &sactions, NULL );
        (void) sigaction( SIGINT,  &sactions, NULL );
        (void) sigaction( SIGQUIT, &sactions, NULL );

        /*
        * Setup mask so we can specify what signals we will ignore.
        */
        (void) sigfillset( &sactions.sa_mask );

       /*
        * We want to ignore everything except those we have set up
        * above so remove those from the list.
        */
        (void) sigdelset( &sactions.sa_mask, SIGTERM );
        (void) sigdelset( &sactions.sa_mask, SIGINT );
        (void) sigdelset( &sactions.sa_mask, SIGQUIT );

       /*
        * Set the mask. Since no other threads have been started,
        * all threads will get this mask.
        */
        (void) thr_sigsetmask( SIG_SETMASK, &sactions.sa_mask, NULL );
}


/*****************************************************************************
**
** Routine: daemon_check_proper_ID
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              exits with an error when the user is not root
**
** Purpose:     Checks user's ID and determines if the user is allowed
**              to execute service. If there are no constraints then this
**              function may be blank.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/

void daemon_check_proper_ID()
{
        /*
        ** Check for root
        */
```



```
        if (geteuid() != E_ROOTUID)
        {
          (void) EDMRestoreEng_logent(
                        __FILE__,__LINE__, LOG_ERR, DAEMON_NOTSUPERUSER, 0,
                        "Must be run as superuser, uid was %d",
                        geteuid());
          exit(1);
        }
}

/*****************************************************************************
**
** Routine: parse_commandline
**
** Inputs:      argc, argv (command line arguments)
**
** Outputs:     None
**
** Return Codes:
**              exits with an error when the user types a bad argument
**
** Purpose:     Parses command line arguments and sets flags. If there
**              are no flags to be set then this function may be empty.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/

void parse_commandline(int argc, char *argv[])
{
int                     opt;            /* Process options   */

        commandlineargs = argv;

        while ((opt = getopt(argc,argv,"dD")) != EOF )
        {
          switch(opt)
          {
            case 'd':
            case 'D':
                G_debug = TRUE;
                debugmode = 1;          /* turn on other debugmode flag */
                break;
            default:
                (void) display_usage( argv[0] );
                exit(1);

          }
        }
}

/*****************************************************************************
**
** Routine: daemon_initialize_logging
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     Do whatever it takes to initialize logging. In the near
**              future this may involve doing something with catalogs or
```

```
**              calling higher level logging functions which encapsulate
**              these things.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/

void
daemon_initialize_logging()
{
        /* Pass in argv[0], the program name */
        (void) esl_log_init(commandlineargs[0]);
}

/*****************************************************************************
**
** Routine: daemon_become_daemon
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              exits with an error code if initialization fails
**
** Purpose:     This function is for doing the forking etc. under UNIX.
**              It is unknown what will be necessary under NT.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/

void
daemon_become_daemon()
{
         char *ptr;
        int  ret = 0;

        /*
         * Strip the path from the program name so we can use it
         * elsewhere.
         */
        ptr = strrchr(commandlineargs[0], '/');
        if (ptr == NULL)
            ptr = commandlineargs[0];
        else
            ptr++;

        /* Change directory to a process specific core directory */
        ret = esl_coredir_setup(ptr);
        if (ret != 0)
        {
             (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
                                MESSAGE_ERR_IN_ESL_COREDIR, errno,
                                "esl_coredir_setup failed" );
            exit(1);
        }

         /*
         **  This is now esl functionality.                  This code does everything necessary
         **  to make this a "real" daemon by detaching from the terminal
         **  changing the process group, closing stdout, stderr, stdin, ...
         */
```



```
/***    if (G_debug == FALSE)
        {
            ret = esl_daemon_startup();
            if (ret != 0)
            {
                fprintf(stderr, "%s: Failed to initialize as daemon.\n",
                        commandlineargs[0]);
                exit(1);
            }
        }   **/
}

/*****************************************************************************
**
** Routine: rpc_init
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              exits with an error code if initialization fails
**
** Purpose:     This function is for doing RPC initialization.
**              For the most part it involves calling the csc routines.
**              This is pretty standard between UNIX and NT.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/

void rpc_init()
{
error_status_t          status;                 /* error status (nbase.h) */
unsigned char    *conn_h;
struct hostent  *hp;
char    *ebuff;
struct utsname name;

/*
** This is here because of HP which may or may not define timeval.
** May be removed when esl_timeval is ported to clients
*/
#ifdef _STRUCT_TIMEVAL
        struct timeval  sleep_interval = {5,0};
                                                /* 5 second sleep interval */
#else
        struct timespec sleep_interval = {5,0};
                                                /* 5 second sleep interval */
#endif

        /* Setup the interface specification for RPC */

        RE_SERVER_IFSPEC(if_spec);

        /*
        * Login as SERVER_PRINCIPAL.  The context of the process
        * will be set to this principal.
        *
        * This process will keep trying to login to DCE if the registery
        * server is unavailable.  Note that under SUN RPC this is a no-op.
        */
        while (TRUE)
        {
```

```
                (void) csc_server_login(RE_SERVER_PRINCIPAL,
                                        RE_SERVER_KEYTAB, &status);

                /* If we succeeded, then exit this loop.  */
                if ( status == error_status_ok )
                {
                        break;
                }
                else    /* Print error message if appropriate. */
                {
                 ebuff = (char *) csc_get_error( status );

                   (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
                                        MESSAGE_NO_LOGIN, 0,
                                        "CSC_SERVER_LOGIN failed: <%d> %s",
                                        status, (
                                                ebuff ? ebuff : "Unknown error"));
                }

                /* If the failure was due to unavailable client,
                * pause and then try again.
                */
                if (status == sec_rgy_server_unavailable)
                {
                        /*
                        * uses sleep when SUNRPC, otherwise uses
                        * pthread call to delay for the specified
                        * time
                        */
                          CSC_SLEEP(sleep_interval);
                          continue;
                }

                /* If we got here, we had a unexpected failure.  */
                (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
                                        MESSAGE_NO_LOGIN, 0,
                                        "The service cannot log in as required");

                exit(1);
        }

        uname(&name);

        hp = gethostbyname(name.nodename);

        if (hp == NULL)
        {
             (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
                                        MESSAGE_GETHOSTBYNAME_FAIL, errno,
                                        "gethostbyname failed" );

             exit(1);
        }

        memcpy((char *) &if_spec.ip_addr, hp -> h_addr, hp->h_length);

         /*
         ** We need to initialize the authorization module before we do
         ** a listen.
         */
         (void)csc_authorization_init(&status);

         if ( status != error_status_ok )
         {
          ebuff = (char *) csc_get_error( status );

            (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
```



```
                                        MESSAGE_NOAUTHORIZATION, 0,
                                        "CSC_AUTHORIZATION_INIT failed: <%d> %s",
                                         status, (ebuff ? ebuff : "Unknown error") );
            exit(1);
         }

        conn_h = calloc(1, CONNECT_HANDLE_SIZE);

        if (conn_h == NULL)
        {
            (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
                                        MESSAGE_NO_MEMORY, 0,
                                        "Failure allocating memory for connection
                                                                        handle");
            exit(1);
        }

         (void) csc_register_private_server_interface(&if_spec,
                                                      0,
                                                      1,
                                                      conn_h,
                                                      &status);

         if ( status != error_status_ok )
         {
          ebuff = (char *) csc_get_error( status );

            (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
                                        MESSAGE_CANNOTREGISTER, 0,
                                        "CSC_REGISTER_SERVER_INTERFACE failed: <%d> %s",
                                         status, (ebuff ? ebuff : "Unknown error") );
            exit(1);
         }

         free(conn_h);
}

/*****************************************************************************
**
** Routine: rpc_run
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     This function is for running the RPC listen.
**              This is pretty standard between UNIX and NT.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/

void rpc_run()
(
error_status_t          status;                 /* error status (nbase.h) */
char  *ebuff;

        /* listen for RPC calls forever. */
         (void) csc_server_listen( rpc_c_listen_max_calls_default, &status );

        ebuff = (char *) csc_get_error( status );
```

```
        /* We don't expect to get here. */
        (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
                                MESSAGE_SERVERLISTEN, 0,
                                "CSC_SERVER_LISTEN failed: <%d> %s",
                                status, (ebuff ? ebuff : "Unknown error") );
}

/*****************************************************************************
**
** Routine: daemon_specific_initialization
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     Do whatever makes this daemon special. In some cases you
**              may want to start a thread or open a socket. Do that here.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/

void
daemon_specific_initialization()
{
int                     status;                 /* error status (nbase.h) */
void                    *statptr;               /* error status (nbase.h) */
int                     ret;
pthread_t               pmtid;
pthread_t               progresstid;
pthread_t               ccrtid;
pthread_t               ccwtid;
time_t    current_time;
char      *ctimebuf;

        ret = CommandAPIInit(&status);
        ret = QuestionAPIInit(&status);
        ret = DrainAPIInit(&status);

        /* Find out what time it is */
        (void) time(&current_time);

        ctimebuf = ctime(&current_time);

        /* Overlay newline with null - buf should always be 26 bytes long */
        ctimebuf[ strlen(ctimebuf) - 1 ] = 0;

        /* Log startup message */
        (void) EDMRestoreEng_logent( __FILE__, __LINE__, LOG_INFO,
                                MESSAGE_STARTUP, 0,
                                        "Restore service %s starting up at %s",
                                commandlineargs[0], ctimebuf );

       /*
        * Start the other threads in the daemon. The main thread
        * becomes the RPC thread. REProcessManager is the
        * entry point for the periodic event thread.
        */
       pthread_create(&pmtid, NULL, REProcessManager, NULL);
       pthread_create(&progresstid, NULL, REProgress, NULL);
       /* pthread_create(&ccrtid, NULL, RestoreSvc_ccr, NULL); */
        pthread_create(&ccwtid, NULL, RestoreSvc_ccw, NULL);
```



```
        rpc_init();
         RestoreSvc_Setup();
        rpc_run();

        pthread_join(pmtid, &statptr);
}

/*****************************************************************************
**
** Routine: daemon_cleanup
**
** Inputs:      None
**
** Outputs:     None
**
** Return Codes:
**              None
**
** Purpose:     Call function which will clean up daemon properly.
**
** Intended caller:  internal only.
**
*****************************************************************************
*/

void
daemon_cleanup()
{
        kill_handler( 0 );
}
```

```
/*
** Copyright 1996,1997 EMC Corporation
*/


/* EDMRestoreEngService.c
 *
 *
 * Mission Statement:  RPC entry points.
 *
 * Primary Data Acted On:
 *
 * Compile-Time Options:
 *
 * Basic idea here:
*/

#if !defined(lint)
static char      RCS_id [] = "@(#)$RCSfile: rpcsvc.c,v $ "
                             "$Revision: 1.0 $ "
                             "$Date: 1997/02/06 20:49:15 $" ;
#endif

#define RAW_NETWORK 0
#define PLUGIN 1



#include <esl/c_portable.h>
#include <esl/inout.h>
#include <util/esl_string.h>

#include <logging/logging.h>
#include <csc/cscomm.h>
#include <errno/e_eb.h>

#include <restore/csc_EDMRestoreEng.h>
#include <restore/restore_engine.h>
#include <restore/REprogmsg.h>

#include <EDMRestoreEngLog.h>
#include <RSLapi.h>
#include <EDMRECommandApi.h>
#include <restore/EDMREProgressApi.h>
#include <EDMREQuestionApi.h>
#include <EDMRENotifyApi.h>

#include <sys/time.h>

/*
 * External prototypes that are defined locally because of header file
 * conflicts between restore_engine.h and restoreRPC.h
 */

void RSTLL_FreeTimelist( struct RSTRPC_time_list **listhead );
void RSTLL_FreeNameList( struct RSTRPC_name_list **listhead );

/*
 * Local Constants:
 */
/* This constant is designed to allow an asynchronous RPC to complete after
 * an interrupt signal is sent, but not allow the canceling RPC to time out */
#define MAX_CANCEL_WAIT_SECS      20

/* This constant is designed to allow the get_restore_feedback RPC to
 * complete quickly after an interrupt signal is sent, if the cancelation
```



```
 * does not take effect immediately.
 */
#define MAX_CANCEL_RESTORE_WAIT_SECS      1

/*
 * Local functions:
 */
static void set_rpc_obj( ulong rpc_id, RE_rpc_objID *rpc_objID );
static RE_errno_ty check_RPC_state( boolean_ty set, int cmd );
static void clear_RPC_state( void );

/*
 * Local static data:
 */
static int        current_rpc_cmd = COMMAND_NONE_ACTIVE;

/*****************************************************************************
**
** Routine:  re_initialize__svc_1
**
** Inputs:   RE_initialize_args * - args for the restore initialize call
**
** Outputs:  None
**
** Return Codes:
**           RE_initialize_result * - result of init function call
**
** Purpose:  Function to create a restore session.
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_initialize_1_svc(IN RE_initialize_args *arg, IN struct svc_req *req )
{
        static RE_status_result argzz;

        setLastRpcTime( );        /* note time of last RPC */
        /* allow multiple calls to Initialize while debugging */
        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)           /* if not idle, trouble */
                ;         /* we weren't idle, reject call */
        else
                argzz.status = RSTSL_Initialize( arg->username);
        if (argzz.status == E_SUCCESS) {
                setGlobalStatus( EDMRE_STATE_BROWSING );
                                                /* after init is browsing */
                clear_RPC_state( );
        }
        else
                setGlobalStatus( EDMRE_STATE_FAILED );
                                                /* without init, we're dead */

        set_rpc_obj( re_initialize, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_get_source_hosts
**
** Inputs:   RE_get_hosts_args * - args for the get source hosts call
**
** Outputs:  None
```

```
 **
 ** Return Codes:
 **          RE_get_hosts_result * - result of get source hosts function call
 **
 ** Purpose:  Function to retrieve the backup client hosts
 **
 ** Intended caller:  RPC call from Restore API client
 ******************************************************************************
 */

RE_get_hosts_result *
re_get_source_hosts_1_svc( IN RE_get_hosts_args *arg, IN struct svc_req *req )
{
        static RE_get_hosts_result argzz;
        static RSTRPC_name_list         *hosts = NULL;

        setLastRpcTime( );      /* note time of last RPC */
        if (hosts)
                RSTLL_FreeNameList( &hosts );   /* free old namelist */

        argzz.cookie = arg->cookie;
        argzz.numEntries = 0;
        argzz.hosts = NULL;

        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)                 /* if not idle, trouble */
                ;               /* we weren't idle, leave hosts=NULL; reject call */
        else
                argzz.status = RSTSL_GetSourceHosts( arg->hostname,
                                                     arg->maxEntries,
                                                     &hosts,
                                                     &argzz.numEntries,
                                                     &argzz.cookie );

        if (argzz.status == E_SUCCESS)
                argzz.hosts = hosts;

        set_rpc_obj( re_get_source_hosts, &argzz.RPCobjID );

        return &argzz;
}

/******************************************************************************
 **
 ** Routine:  re_get_destination_hosts
 **
 ** Inputs:  RE_get_hosts_args  * - args for the RPC call
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **          RE_get_hosts_result * - result of RPC function call
 **
 ** Purpose:  Function to retrive the names of the possible restore target
                                                                        hosts
 **
 ** Intended caller:  Internal Only.
 ******************************************************************************
 */

RE_get_hosts_result *
re_get_destination_hosts_1_svc(
                        IN RE_get_hosts_args *arg, IN struct svc_req *req )
{
        static RE_get_hosts_result argzz;
        static RSTRPC_name_list         *hosts = NULL;
```



```
        setLastRpcTime( );      /* note time of last RPC */
        if (hosts)
        {
                RSTLL_FreeNameList( &hosts );   /* free old namelist */
        }

        argzz.cookie = arg->cookie;
        argzz.numEntries = 0;
        argzz.hosts = NULL;

        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)                 /* if not idle, trouble */
                ;       /* we weren't idle, leave hosts=NULL; reject call */
        else {
                argzz.status = RSTSL_GetDestinationHosts( arg->maxEntries,
                                                          &hosts,
                                                          &argzz.numEntries,
                                                          &argzz.cookie );

                if (E_SUCCESS == argzz.status)
                        argzz.hosts = hosts;
        }

        set_rpc_obj( re_get_destination_hosts, &argzz.RPCobjID );

        return &argzz;
}

/******************************************************************************
 **
 ** Routine:  re_get_top_level_objects
 **
 ** Inputs:   RE_get_top_level_objects_args * - args for the top level objs
                                                                        call
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **          RE_get_top_level_objects_result * - result of function call
 **
 ** Purpose:  Function to retrive the top level objects (
                                                        workitem, workitem sets)
 **
 ** Intended caller:  Internal Only.
 ******************************************************************************
 */

RE_get_top_level_objects_result *
re_get_top_level_objects_1_svc( IN RE_get_top_level_objects_args *arg,
                                IN struct svc_req *req )
{
        static RE_get_top_level_objects_result argzz;
        static short lastNumEntries = 0;
        RSTRPC_tlo_list                 *topListPtr;
        RSTRPC_top_level_obj    *tloPtr;

        setLastRpcTime( );      /* note time of last RPC */
        /* free last call's output: */
        if (lastNumEntries) {
                xdr_free( xdr_RE_get_top_level_objects_result, (
                                                                char *)&argzz);
                lastNumEntries = 0;
        }

        argzz.cookie = arg->cookie;
```

```
        argzz.numEntries = 0;
        argzz.topLevelObjs = NULL;

        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)          /* if not idle, trouble */
                ;          /* we weren't idle, leave hosts=NULL; reject call */
        else
                argzz.status = RSTSL_GetTopLevelObjects( arg->sourceHost,
                                                         arg->maxEntries,
                                                         &argzz.topLevelObjs,
                                                         &argzz.numEntries,
                                                         &argzz.cookie,
                                                         PLUGIN);

        lastNumEntries =  argzz.numEntries;

/* Fix returned objects to avoid null string pointers for RPC :  */
        topListPtr = argzz.topLevelObjs;
        while (topListPtr)
        {
                tloPtr = topListPtr->tlo;
                if (!tloPtr->root.objName)
                        tloPtr->root.objName = esl_strdup( "" );
                if (!tloPtr->root.objTypeString)
                        tloPtr->root.objTypeString = esl_strdup( "" );
                if (!tloPtr->fileSpec)
                        tloPtr->fileSpec = esl_strdup( "" );
                if (!tloPtr->templateName)
                        tloPtr->templateName = esl_strdup( "" );
                if (!tloPtr->hostname)
                        tloPtr->hostname = esl_strdup( "" );
                if (!tloPtr->wiBIC)
                        tloPtr->wiBIC = esl_strdup( "" );
#if 0   /* this might cause problem: 0 length, 1 char buffer */
                if (!tloPtr->appData.appData_val)
                        tloPtr->appData.appData_val = esl_strdup( "" );
#endif
                topListPtr = topListPtr->next;
        }

        set_rpc_obj( re_get_top_level_objects, &argzz.RPCobjID );

        return &argzz;
}
/*****************************************************************************
**
** Routine:  re_get_all_top_level_objects
**
** Inputs:   RE_get_top_level_objects_args * - args for the top level objs
                                                                        call
**
** Outputs:  None
**
** Return Codes:
**           RE_get_top_level_objects_result * - result of function call
**
** Purpose:  Function to retrive the top level objects (
                                                        workitem, workitem sets)
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_get_top_level_objects_result *
re_get_all_top_level_objects_1_svc( IN RE_get_top_level_objects_args *arg,
                                    IN struct svc_req *req )
{
```



```
        static RE_get_top_level_objects_result argzz;
        static short lastNumEntries = 0;
        RSTRPC_tlo_list                 *topListPtr;
        RSTRPC_top_level_obj    *tloPtr;

        setLastRpcTime( );      /* note time of last RPC */
        /* free last call's output: */
        if (lastNumEntries) {
                xdr_free( xdr_RE_get_top_level_objects_result, (
                                                                char *)&argzz);
                lastNumEntries = 0;
        }

        argzz.cookie = arg->cookie;
        argzz.numEntries = 0;
        argzz.topLevelObjs = NULL;

        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)          /* if not idle, trouble */
                ;          /* we weren't idle, leave hosts=NULL; reject call */
        else
                argzz.status = RSTSL_GetTopLevelObjects( arg->sourceHost,
                                                         arg->maxEntries,
                                                         &argzz.topLevelObjs,
                                                         &argzz.numEntries,
                                                         &argzz.cookie,
                                                         RAW_NETWORK);

        lastNumEntries =  argzz.numEntries;

/* Fix returned objects to avoid null string pointers for RPC :  */
        topListPtr = argzz.topLevelObjs;
        while (topListPtr)
        {
                tloPtr = topListPtr->tlo;
                if (!tloPtr->root.objName)
                        tloPtr->root.objName = esl_strdup( "" );
                if (!tloPtr->root.objTypeString)
                        tloPtr->root.objTypeString = esl_strdup( "" );
                if (!tloPtr->fileSpec)
                        tloPtr->fileSpec = esl_strdup( "" );
                if (!tloPtr->templateName)
                        tloPtr->templateName = esl_strdup( "" );
                if (!tloPtr->hostname)
                        tloPtr->hostname = esl_strdup( "" );
                if (!tloPtr->wiBIC)
                        tloPtr->wiBIC = esl_strdup( "" );
#if 0   /* this might cause problem: 0 length, 1 char buffer */
                if (!tloPtr->appData.appData_val)
                        tloPtr->appData.appData_val = esl_strdup( "" );
#endif
                topListPtr = topListPtr->next;
        }

        set_rpc_obj( re_get_top_level_objects, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_get_restorable_objects_start
**
** Inputs:   RE_get_restorable_objects_start_args *
**
** Outputs:  None
**
```

```
** Return Codes:
**         RE_get_restorable_objects_start_result *
**
** Purpose:  Function to start the retrieval of the child objects of the
**       specified parent object. The caller specifies the parent object
**       and whether or not to include bad files.
**
** Intended caller:  RPC call from Restore API client
*****************************************************************************
*/

RE_get_restorable_objects_start_result *
re_get_restorable_objects_start_1_svc(
                                IN RE_get_restorable_objects_start_args *arg,
                                  IN struct svc_req *req )

{
        static RE_get_restorable_objects_start_result   argzz;
        RE_get_restorable_objects_start_args            *cmd_args;

        int     status;

        setLastRpcTime( );      /* note time of last RPC */
        cmd_args = calloc( 1, sizeof(RE_get_restorable_objects_start_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                        "Cannot malloc RE_get_restorable_objects_start_args" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no RPC is in progress */
        else if (E_SUCCESS != (argzz.status = check_RPC_state( TRUE,
                                        COMMAND_GET_RESTORABLE_OBJECTS )) )
                ;       /* just return failure status */
        else
        {
                cmd_args->parentObj = arg->parentObj;
                /* change null string template name to NULL ptr */
if (cmd_args->parentObj->objLevel == RSTRPC_tlo_type
 && cmd_args->parentObj->RE_restorable_obj_u.tloInfo->templateName
 && !strlen( cmd_args->parentObj->RE_restorable_obj_u.tloInfo->templateName) )
                {
        free( cmd_args->parentObj->RE_restorable_obj_u.tloInfo->templateName);
        cmd_args->parentObj->RE_restorable_obj_u.tloInfo->templateName = NULL;
                }
                arg->parentObj = NULL;
                cmd_args->cookie =  arg->cookie;
                cmd_args->maxEntries = arg->maxEntries;
                cmd_args->allowBadFiles = arg->allowBadFiles;

                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                              "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand(
                                COMMAND_GET_RESTORABLE_OBJECTS, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                              "PushCommand failed");
```



```
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                        argzz.status = E_SUCCESS;
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_get_restorable_objects_start_args,
                                  (char *)cmd_args);
                        free( cmd_args );
                }
        }
        set_rpc_obj( re_get_restorable_objects_start, &argzz.RPCobjID );
        return &argzz;
}


/*****************************************************************************
**
** Routine:  re_get_restorable_objects_output
**
** Inputs:   RE_get_restorable_objects_output_args *
**
** Outputs:  None
**
** Return Codes:
**         RE_get_restorable_objects_output_result *
**
** Purpose:  Function to test for completion of the
**       re_get_restorable_objects_start_1 RPC call, and retrieve some or all
**       of its output.
**
** Intended caller:  RPC call from Restore API client
*****************************************************************************
*/

RE_get_restorable_objects_output_result *
re_get_restorable_objects_output_1_svc(
                                IN RE_get_restorable_objects_output_args *arg,
                                IN struct svc_req *req )
{
        static RE_get_restorable_objects_output_result  argzz;
        static RE_get_restorable_objects_output_result  *outarg = NULL;
        int     result, cmd, status;

        setLastRpcTime( );      /* note time of last RPC */
        if (outarg)
        {       /* free last results */
                xdr_free( xdr_RE_get_restorable_objects_output_result,
                          (char *)outarg );
                free( outarg );
                outarg = NULL;
        }

        else
        {       /* init static output struct for errors (
                                                        lst time & aft errs */
                argzz.numEntries = 0;
                argzz.cookie = 0;
                argzz.childrenObjs = NULL;
        }
```

```
        /* make sure this RPC is in progress */
        if (E_SUCCESS != (argzz.status = check_RPC_state( FALSE,
                                        COMMAND_GET_RESTORABLE_OBJECTS )) )
                ;       /* just return failure status */

        /* test for completion of processing: later use real flag */
        else if (PopResult( -1, &result, &cmd, &status) )
        {
                if (status == COMMAND_RECORD_GET_FAILED)
                {
                        argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                } else {
                        /* log error, clean up, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        status, 0, "PopResult failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
        }
        else if (cmd != COMMAND_GET_RESTORABLE_OBJECTS)
        {
                /* log error, clean up, return error  */
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        MESSAGE_INVALID_COMMAND, 0,
                        "PopResult mismatch: got %d command, expected %d\n",
                                        cmd, COMMAND_GET_RESTORABLE_OBJECTS);
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (result != COMMAND_RESULT_SUCCESS)
        {
               EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                MESSAGE_FAILURE_DOING_ASYNC_RPC , 0,
                                "RPC failure in process manager thread" );
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (PopRpcOutput( (void **)&outarg, &status) )
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status,
                                        0, "PopRpcOutput failure");
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else
        {       /* return popped results struct */
            set_rpc_obj( re_get_restorable_objects_output, &outarg->RPCobjID);
            clear_RPC_state( );
                                        /* indicate process mgr idle */
            return outarg;
        }

        /* return static result struct on errors */
        set_rpc_obj( re_get_restorable_objects_output, &argzz.RPCobjID );
        if (argzz.status == EP_RB_RECOVER_SERVERFAIL)
            clear_RPC_state( );
                                /* indicate process mgr idle on fatals */
        return &argzz;
}


/*****************************************************************************
**
** Routine:  re_mark_object
**
** Inputs:   RE_mark_object_args *
**
** Outputs:  None
**
** Return Codes:
```



```
**          RE_mark_object_result *
**
** Purpose:  Function to start the marking process for a user restorable
**       object and, optionally, for its descendants.
**
** Intended caller:  RPC call from Restore API client
*****************************************************************************
*/

RE_mark_object_result *
re_mark_object_1_svc( IN RE_mark_object_args *arg, IN struct svc_req *req )
{
        static RE_mark_object_result    argzz;
        RE_mark_object_args             *cmd_args;

        int     status;

        cmd_args = calloc( 1, sizeof(RE_mark_object_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        MESSAGE_NO_MEMORY, errno,
                        "Cannot malloc RE_mark_object_args" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no rpc is in progress */
        else if ( (argzz.status = check_RPC_state(
                                                TRUE, COMMAND_MARK_OBJECT ))
          != E_SUCCESS )
                ;       /* just return failure status */

        else
        {
                ClearRpcCancelFlag( );  /* reset cancel flag */
                ClearProgressValue( );  /* reset progress count */

                cmd_args->thisObj = arg->thisObj;
                arg->thisObj = NULL;    /* so RPC stuff wont free it */
                cmd_args->backupTime =  arg->backupTime;
                cmd_args->allowBadFiles = arg->allowBadFiles;
                cmd_args->descend = arg->descend;

                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        status, 0,
                                                "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand( COMMAND_MARK_OBJECT, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        status, 0,
                                                "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                        argzz.status = E_SUCCESS;
        }

        if (argzz.status != E_SUCCESS)
```

```
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_mark_object_args, (char *)cmd_args );
                        free( cmd_args );
                }
        }

        set_rpc_obj( re_mark_object, &argzz.RPCobjID );
        return &argzz;
}


/*****************************************************************************
**
** Routine:  re_get_mark_results
**
** Inputs:   RE_get_mark_results_args *
**
** Outputs:  None
**
** Return Codes:
**       RE_get_mark_results_result *
**
** Purpose:  Function to test for completion of, or interrupt, the
**       re_mark_object RPC call, and retrieve its output.
**
** Intended caller:  RPC call from Restore API client
*****************************************************************************
*/

RE_get_mark_results_result *
re_get_mark_results_1_svc( IN RE_get_mark_results_args *arg,
                           IN struct svc_req *req )
{
        static RE_get_mark_results_result       argzz;
        static RE_get_mark_results_result       *outarg = NULL;
        int     result, cmd, status;

        setLastRpcTime( );      /* note time of last RPC */
        if (outarg)
        {       /* free last results */
                xdr_free( xdr_RE_get_mark_results_result, (char *)outarg );
                free( outarg );
                outarg = NULL;
        }

        else
        {       /* init static output struct for errors (
                                                        lst time & aft errs */
                argzz.badFileCount = 0;
                argzz.permDenyFileCount = 0;
                argzz.dirMarkCount = 0;
                argzz.fileMarkCount = 0;
                argzz.otherMarkCount = 0;
        }

        /* make sure mark is in progress */
        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_MARK_OBJECT ))
          != E_SUCCESS )
                ;       /* just return failure status */
        /* test for completion of processing; later use real flag */
        else if (PopResult( -1, &result, &cmd, &status) )
        {
                if (status == COMMAND_RECORD_GET_FAILED)
                {
                        if (arg->interrupt)
```



```
                {       /* signal cancel, wait till done */
                        SetRpcCancelFlag( );
                        if (PopResult( MAX_CANCEL_WAIT_SECS, &result,
                                        &cmd, &status) )
                                /* if no result, error */
                                argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        }
                        else {
                                argzz.fileMarkCount = ReadProgressValue( );
                                argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                        }
                }

                else {
                        /* log error, clean up, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        status, 0, "PopResult failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
        }

        if (argzz.status != E_SUCCESS)
                ;       /* fall thru to error return logic */

        else if (cmd != COMMAND_MARK_OBJECT)
        {
                /* log error, clean up, return error  */
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                MESSAGE_INVALID_COMMAND, 0,
                        "PopResult mismatch: got %d command, expected %d\n",
                                cmd, COMMAND_MARK_OBJECT);
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (result != COMMAND_RESULT_SUCCESS)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                MESSAGE_FAILURE_DOING_ASYNC_RPC, 0,
                                "RPC failure in process manager thread" );
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (PopRpcOutput( (void **)&outarg, &status) )
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status,
                                        0, "PopRpcOutput failure");
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else
        {       /* return popped results struct */
            set_rpc_obj( re_get_mark_results, &outarg->RPCobjID);
            clear_RPC_state( );
                                                /* indicate process mgr idle */
            return outarg;
        }

        set_rpc_obj( re_get_mark_results, &argzz.RPCobjID );
        if (argzz.status == EP_RB_RECOVER_SERVERFAIL)
            clear_RPC_state( );
                                        /* indicate process mgr idle on fatals */
        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_unmark_object_1
**
** Inputs:   RE_unmark_object_args * - args for the RPC call
```

```
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **          RE_mark_object_result * - result of RPC function call
 **
 ** Purpose:  Function to unmark objects for restoral
 **
 ** Intended caller:  Internal Only.
 ********************************************************************************
 */

RE_mark_object_result *
re_unmark_object_1_svc(IN RE_unmark_object_args *arg, IN struct svc_req *req )
{
        static RE_mark_object_result    argzz;
        RE_unmark_object_args           *cmd_args;
        int                             status;

        setLastRpcTime( );      /* note time of last RPC */
        cmd_args = calloc( 1, sizeof(RE_unmark_object_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                        "Cannot malloc RE_unmark_object_args" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no rpc is in progress */
        else if ( (argzz.status = check_RPC_state(
                                                TRUE, COMMAND_UNMARK_OBJECT ))
          != E_SUCCESS )
                ;       /* just return failure status */

        else
        {
                ClearRpcCancelFlag( );  /* reset cancel flag */
                ClearProgressValue( );  /* reset progress count */

                cmd_args->thisObj = arg->thisObj;
                arg->thisObj = NULL;    /* so RPC stuff wont free it */
                cmd_args->backupTime =  arg->backupTime;
                cmd_args->badFilesOnly = arg->badFilesOnly;
                cmd_args->descend = arg->descend;

                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                              "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand( COMMAND_UNMARK_OBJECT, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                              "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                        argzz.status = E_SUCCESS;
```



```
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_unmark_object_args, (
                                                        char *)cmd_args );
                        free( cmd_args );
                }
        }


        set_rpc_obj( re_unmark_object, &argzz.RPCobjID );

        return &argzz;
}
/* re_unmark_object_1 */

/********************************************************************************
 **
 ** Routine:  re_get_unmark_results
 **
 ** Inputs:  RE_get_mark_results_args  * - args for the RPC call
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **          RE_get_unmark_results_result * - result of RPC function call
 **
 ** Purpose:  Function to test for completion of the unmark request
 **
 ** Intended caller:  Internal Only.
 ********************************************************************************
 */

RE_get_unmark_results_result *
re_get_unmark_results_1_svc(IN RE_get_mark_results_args *arg,
                            IN struct svc_req *req )
{
        static RE_get_unmark_results_result     argzz;
        static RE_get_unmark_results_result     *outarg = NULL;
        int     result, cmd, status;

        setLastRpcTime( );      /* note time of last RPC */

        if (outarg)
        {       /* free last results */
                xdr_free( xdr_RE_get_unmark_results_result, (char *)outarg );
                free( outarg );
                outarg = NULL;
        }

        else
        {       /* init static output struct for errors (
                                                        1st time & aft errs */
                argzz.badFileCount = 0;
                argzz.dirMarkCount = 0;
                argzz.fileMarkCount = 0;
                argzz.otherMarkCount = 0;
        }

        /* make sure unmark is in progress */
        if ( (argzz.status = check_RPC_state(
                                                FALSE, COMMAND_UNMARK_OBJECT ))
          != E_SUCCESS )
                ;       /* just return failure status */
```

```
        /* test for completion of processing: later use real flag */
        else if (PopResult( 1, &result, &cmd, &status) )
        {
                if (status == COMMAND_RECORD_GET_FAILED)
                {
                        if (arg->interrupt)
                        {       /* signal cancel, wait till done */
                            SetRpcCancelFlag( );
                            if (PopResult( MAX_CANCEL_WAIT_SECS, &result,
                                           &cmd, &status) )
                                /* if no result, error */
                                argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        }
                        else {
                                argzz.fileMarkCount = ReadProgressValue( );
                                argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                        }
                }

                else {
                        /* log error, clean up, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0, "PopResult failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
        }

        if (argzz.status != E_SUCCESS)
                ;       /* fall thru to error return logic */

        else if (cmd != COMMAND_UNMARK_OBJECT)
        {
                /* log error, clean up, return error  */
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_INVALID_COMMAND, 0,
                        "PopResult mismatch: got %d command, expected %d\n",
                                      cmd, COMMAND_UNMARK_OBJECT);
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (result != COMMAND_RESULT_SUCCESS)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_FAILURE_DOING_ASYNC_RPC, 0,
                                "RPC failure in process manager thread" );
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (PopRpcOutput( (void **)&outarg, &status) )
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status,
                                      0, "PopRpcOutput failure");
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else
        {       /* return popped results struct */
            set_rpc_obj( re_get_unmark_results, &outarg->RPCobjID);
            clear_RPC_state( );
                                                /* indicate process mgr idle */
            return outarg;
        }

        set_rpc_obj( re_get_unmark_results, &argzz.RPCobjID );
        if (argzz.status == EP_RB_RECOVER_SERVERFAIL)
            clear_RPC_state( );
                                        /* indicate process mgr idle on fatals */

        return &argzz;
```



```
}
    /* re_get_unmark_results_1 */

/*****************************************************************************
**
** Routine:  re_submit
**
** Inputs:  RE_submit_args  * - args for the RPC call
**
** Outputs: RE_status_result * - result of RPC function call
**
** Purpose:  Function to prepare for the restore of the currently marked
**       objects
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_submit_1_svc( IN RE_submit_args *arg,
                 IN struct svc_req *req )

{
        static RE_status_result         argzz;
        RE_submit_args          *cmd_args;
        int                     status;

        setLastRpcTime( );      /* note time of last RPC */

        cmd_args = calloc( 1, sizeof(RE_submit_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                        "Cannot malloc RE_submit_args" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no rpc is in progress */
        else if ( (argzz.status = check_RPC_state( TRUE, COMMAND_SUBMIT ))
          != E_SUCCESS )
                ;       /* just return failure status */

        else
        {
                ClearRpcCancelFlag( );  /* reset cancel flag */
                ClearProgressValue( );  /* reset progress count */

                cmd_args->hostname = esl_strdup( arg->hostname );
                cmd_args->directory = esl_strdup( arg->directory );
                cmd_args->overwritePolicy =  arg->overwritePolicy;
                cmd_args->inPlace = arg->inPlace;
                cmd_args->transport = arg->transport;
                cmd_args->submitObjectID = arg->submitObjectID;
                cmd_args->socketClientName = esl_strdup(
                                                arg->socketClientName);
                cmd_args->socketPort= arg->socketPort;
                cmd_args->mapFile_env = esl_strdup(arg->mapFile_env);
                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand( COMMAND_SUBMIT, &status) )
```

```
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                                status, 0,
                                                "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                        argzz.status = E_SUCCESS;
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_submit_args, (char *)cmd_args );
                        free( cmd_args );
                }
        }

    set_rpc_obj( re_submit, &argzz.RPCobjID );

    return &argzz;
}

/*****************************************************************************
**
** Routine:  re_get_submit_results
**
** Inputs:  RE_get_submit_results_args  * - args for the RPC call
**
** Outputs: RE_get_submit_results_output * - result of RPC function call
**
** Purpose:  Function to test for completion of the previously started submit
**           operation.
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_get_submit_results_output *
re_get_submit_results_1_svc( IN RE_get_submit_results_args *arg,
                        IN struct svc_req *req )

{
        static RE_get_submit_results_output     argzz;
        static RE_get_submit_results_output     *outarg = NULL;
        int     result, cmd, status;

        setLastRpcTime( );      /* note time of last RPC */

        if (outarg)
        {       /* free last results */
                xdr_free( xdr_RE_get_submit_results_output, (char *)outarg );
                free( outarg );
                outarg = NULL;
        }

        else
        {       /* init static output struct for errors (
                                                        1st time & aft errs */
                argzz.submitObjectID = 0;
                argzz.objectsDone = 0;
        }

        /* make sure submit is in progress */
```



```
        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_SUBMIT ))
          != E_SUCCESS )
                ;       /* just return failure status */
        /* test for completion of processing: later use real flag */
        else if (PopResult( -1, &result, &cmd, &status) )
        {
                if (status == COMMAND_RECORD_GET_FAILED)
                {
                        if (arg->interrupt)
                        {       /* signal cancel, wait till done */
                            SetRpcCancelFlag( );
                                if (PopResult( MAX_CANCEL_WAIT_SECS, &result,
                                                &cmd, &status) )
                                    /* if no result, error */
                                    argzz.status = EP_RB_RECOVER_SERVERFAIL;
                                }
                                else {
                                        argzz.objectsDone = ReadProgressValue( );
                                        argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                                }
                        }

                        else {
                                /* log error, clean up, return error  */
                                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                                        status, 0, "PopResult failed");
                                argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        }
                }

                if (argzz.status != E_SUCCESS)
                        ;       /* fall thru to error return logic */

                else if (cmd != COMMAND_SUBMIT)
                {
                        /* log error, clean up, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                                MESSAGE_INVALID_COMMAND, 0,
                                "PopResult mismatch: got %d command, expected %d\n",
                                                cmd, COMMAND_SUBMIT);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else if (result != COMMAND_RESULT_SUCCESS)
                {
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                                MESSAGE_FAILURE_DOING_ASYNC_RPC, 0,
                                        "RPC failure in process manager thread" );
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else if (PopRpcOutput( (void **)&outarg, &status) )
                {
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status,
                                                0, "PopRpcOutput failure");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else
                {       /* return popped results struct */
                    set_rpc_obj( re_get_submit_results, &outarg->RPCobjID);
                    clear_RPC_state( );
                                                        /* indicate process mgr idle */
                    return outarg;
                }

                set_rpc_obj( re_get_submit_results, &argzz.RPCobjID );
                if (argzz.status == EP_RB_RECOVER_SERVERFAIL)
                    clear_RPC_state( );
```

```
                                        /* indicate process mgr idle on fatals */
        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_start_1
**
** Inputs:   RE_start_args * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_status_result * - result of RPC function call
**
** Purpose:  Function to start the restore
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_start_1_svc(IN RE_start_args *arg, IN struct svc_req *req )
{
        static RE_status_result argzz;
        RE_start_args           *cmd_args;
        int                     status;

        setLastRpcTime( );      /* note time of last RPC */

        cmd_args = calloc( 1, sizeof(RE_start_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                                      "Cannot malloc RE_start_args" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no rpc is in progress */
        else if ( (argzz.status = check_RPC_state( TRUE, COMMAND_START ))
          != E_SUCCESS )
                ;       /* just return failure status */

        else
        {
              purgeProgress();
               ClearRpcCancelFlag( );  /* reset cancel flag */
               ClearProgressValue( );  /* reset progress count */

               cmd_args->submitObjectID = arg->submitObjectID;

               if (PushRpcInput( (void *)cmd_args, &status) )
               {
                       /* log error, return error  */
                       EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                             status, 0,
                                             "PushRpcInput failed");
                       argzz.status = EP_RB_RECOVER_SERVERFAIL;
                       clear_RPC_state( );     /* indicate idle on fatals */
               }
               else if (PushCommand( COMMAND_START, &status) )
               {
               /* log error, clean up input queue, return error  */
                       EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                             status, 0,
```



```
                                             "PushCommand failed");
                       PopRpcInput( (void **)&cmd_args, &status);
                       argzz.status = EP_RB_RECOVER_SERVERFAIL;
                       clear_RPC_state( );     /* indicate idle on fatals */
               }
               else
               {
                       setExternalStatus( RE_STATE_STARTING );
                       argzz.status = E_SUCCESS;
               }
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_start_args, (char *)cmd_args );
                        free( cmd_args );
                }
        }

        set_rpc_obj( re_start, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_get_restore_feedback
**
** Inputs:  RE_get_restore_feedback_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_get_restore_feedback_result * - result of RPC function call
**
** Purpose:  Function to determine the state of an ongoing restore
**       specified time.
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_get_restore_feedback_result *
re_get_restore_feedback_1_svc(IN RE_get_restore_feedback_args *arg,
                              IN struct svc_req *req )
{
        static RE_get_restore_feedback_result   argzz;
        RE_status_result        *outarg = NULL;
        static RE_Notification          *notify = NULL;
        static long             lasttime = 0;
        int     result, cmd, status, ret = 0;
        struct timeval  timeofday;
        void    *dummy = NULL;

        setLastRpcTime( );      /* note time of last RPC */

        /* init static output struct for progress */
        if (NULL != notify)     /* release old feedback */
                xdr_free( xdr_RE_get_restore_feedback_result, (
                                                        char *)&argzz );
        memset( &argzz, 0, sizeof(RE_get_restore_feedback_result) );
        if (NULL == (notify = calloc( 1, sizeof(RE_Notification) )))
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
```

```
                                "Cannot malloc RE_Notification" );
            argzz.status = EP_RB_RECOVER_NOMEM;

            set_rpc_obj( re_get_restore_feedback, &argzz.RPCobjID );

            return &argzz;
        }

        /* make sure restore (start) is in progress */
        if ( (argzz.status = check_RPC_state(
                                    FALSE, COMMAND_START )) == E_SUCCESS )
        {
            /* test for completion of processing: later use real flag */
            if ( (ret = PopResult( -1, &result, &cmd, &status)) != 0 )
            {
                if (status == COMMAND_RECORD_GET_FAILED)
                {   /* set cancel if requested */
                    if (arg->quit_restore)
                    {
                        SetRpcCancelFlag( );
                        if ( (ret = PopResult( MAX_CANCEL_RESTORE_WAIT_SECS,
                                               &result, &cmd,
                                                        &status)) != 0 )
                        {
                            /* if no result, user must keep trying */
                           argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                        }
                        else ( /* result popped, leave E_SUCCESS to */
                               /* update (final) stats below */
                        }
                    }
                    else        /* no cancel and not done already */
                    {
                        argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                    }
                }
                else {
                    /* log error, clean up, return error  */
                    EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                          status, 0, "PopResult failed");
                    argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
            }

            if (ret == 0)
            {
                if (cmd != COMMAND_START)
                {
                    /* log error, clean up, return error  */
                    EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                          MESSAGE_INVALID_COMMAND, 0,
                            "PopResult mismatch: got %d command,
                                                    expected %d\n",
                                          cmd, COMMAND_START);
                    argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else if (result != COMMAND_RESULT_SUCCESS)
                {
                   EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                         MESSAGE_FAILURE_DOING_ASYNC_RPC,
                                                                    0,
                                  "RPC failure in process manager thread"
                                                                    );
                    argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
```



```
                if (PopRpcOutput( (void **)&outarg, &status) )
                {
                        EDMRestoreEng_logent(
                                        __FILE__, __LINE__, LOG_ERR, status,
                                                0, "PopRpcOutput failure");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else
                {
                        argzz.status = outarg->status;
                        xdr_free( xdr_RE_status_result, (char *)outarg );
                        free( outarg );
                }

                clear_RPC_state( );
                                                /* indicate process mgr idle */
                lasttime = 0;   /* in case multiple starts possible later */
                setGlobalStatus (EDMRE_STATE_BROWSING);
                                                        /* back to browsing */
            }

            if (argzz.status == EP_RB_RECOVER_SERVERFAIL) {
                clear_RPC_state( );
                                                /* indicate process mgr idle on fatals */
                lasttime = 0;   /* in case multiple starts possible later */
                setGlobalStatus (EDMRE_STATE_BROWSING);
                                                        /* back to browsing */
            }
        }

        gettimeofday( &timeofday, dummy );
                                                /* for time of getRestoreStatus */

        if (0 != getRestoreStatus( lasttime, &argzz.rstStats, &status ))
        {
                /* log error, continue */
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status, 0,
                                      "getRestoreStatus failed");
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }

       if (argzz.status == EP_RB_RECOVER_RPC_INCOMPLETE)
        {
                lasttime = timeofday.tv_sec - 120;
        }

        ret = PopNotifications( notify, &status );

        if (ret == 0)
        {
            argzz.notify = notify;
        }

        set_rpc_obj( re_get_restore_feedback, &argzz.RPCobjID );

        return &argzz;
}
        /* end of re_get_restore_feedback_1 */

/*****************************************************************************
**
** Routine:  re_get_question
**
** Inputs:  RE_null_args  * - args for the RPC call (none)
**
** Outputs:  None
```

```
**
** Return Codes:
**          RE_get_question_result * - result of RPC function call
**
** Purpose:  Function to retrieve a restore execution query
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_get_question_result *
re_get_question_1_svc( IN RE_null_args *arg, IN struct svc_req *req )
{
        static RE_get_question_result argzz;
        static Question               question;
        int     result, status;

        setLastRpcTime( );      /* note time of last RPC */

        argzz.query = NULL;     /* init response structure */
        /* dont free last question - its owned by process thread.
                                                        This is copy*/
        memset( &question, 0, sizeof(Question) );

        /* make sure restore (start) is in progress */
        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_START ))
          != E_SUCCESS )
                ;       /* just return failure status */

        else if (getExternalStatus() != RE_STATE_STOPPED)
        {       /* not awaiting answer, either user error or aborted */
                argzz.status = EP_RB_RECOVER_INVALOP;
        }

        /* in proper state: fetch question from question queue */

        else if ( 0 != (result = PopQuestion( 1, &question, &status ) ) )
        {       /* dequeue question failed -- log error, continue */
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status, 0,
                                        "PopQuestion failed");
                if (
                status == QUESTION_RECORD_GET_FAILED) /* assume user wrong */
                        argzz.status = EP_RB_RECOVER_INVALOP;
                else    /* internal error */
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else
                argzz.query = &question;
                                                /* return question structure */
        set_rpc_obj( re_get_question, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_set_user_answer
**
** Inputs:  RE_set_user_answer_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_status_result * - result of RPC function call
**
```



```
** Purpose:  Function to return the user response to a restore execution query
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_set_user_answer_1_svc(IN RE_set_user_answer_args *arg,
                         IN struct svc_req *req )
{
        static RE_status_result argzz;
        int     status;

        setLastRpcTime( );      /* note time of last RPC */

        /* make sure restore (start) is in progress */
        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_START ))
          != E_SUCCESS )
                ;       /* just return failure status */

        else if (getExternalStatus() != RE_STATE_STOPPED)
        {       /* not awaiting answer, either user error or aborted */
                argzz.status = EP_RB_RECOVER_INVALOP;
        }

        /* in proper state: push response on answer queue */

        else if ( PushAnswer( &arg->answers, &status ) )
        {       /* enqueue failed -- log error, continue */
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status, 0,
                                        "PushAnswer failed");
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }

        else
        {       /* restore external state to proper phase */
                if ( EDMRE_STATE_PREPHASE == getGlobalStatus(NULL) )
                        setExternalStatus(RE_STATE_PREPHASE);
                else
                        setExternalStatus(RE_STATE_POSTPHASE);
                /* clear answer list pointer,
                                                since its now on answer queue */
                arg->answers.firstanswer = NULL;
                                                        /* so only freed once */
        }

        set_rpc_obj( re_set_user_answer, &argzz.RPCobjID );

        return &argzz;
}


/*****************************************************************************
**
** Routine:  re_get_top_level_templates_1
**
** Inputs:   RE_get_top_level_templates_args * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_get_top_level_templates_result * - result of RPC function call
**
** Purpose:  Function to retrieve templates configured for the current top
                                                                    level
**          backup object.
```

```
 **
 ** Intended caller:  Internal Only.
 ****************************************************************************

 */

RE_get_top_level_templates_result *
re_get_top_level_templates_1_svc( IN RE_get_top_level_templates_args *arg,
                                  IN struct svc_req *req )
{
        static RE_get_top_level_templates_result argzz;
        static short    lastNumEntries = 0;

        setLastRpcTime( );      /* note time of last RPC */

        /* free last call's output: */
        if (lastNumEntries) {
                xdr_free( xdr_RE_get_top_level_templates_result, (
                                                        char *)&argzz);
                lastNumEntries = 0;
        }

        argzz.cookie = arg->cookie;
        argzz.numEntries = 0;
        argzz.templates = NULL;

        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, leave templates=NULL;
                                                        reject call */
        else {
                argzz.status = RSTSL_GetTopLevelTemplates( arg->topLevelObj,
                                                arg->maxEntries,
                                                &argzz.templates,
                                                &argzz.numEntries,
                                                &argzz.cookie );
                lastNumEntries = argzz.numEntries;
        }

        set_rpc_obj( re_get_top_level_templates, &argzz.RPCobjID );

        return &argzz;
}

/****************************************************************************
 **
 ** Routine:  re_get_current_template
 **
 ** Inputs:  RE_null_args  * - args for the RPC call (none)
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **          RE_get_current_template_result * - result of RPC function call
 **
 ** Purpose:  Function to retrieve trhe currently selected template name
 **
 ** Intended caller:  Internal Only.
 ****************************************************************************
 */

RE_get_current_template_result *
re_get_current_template_1_svc(IN RE_null_args *arg,
                              IN struct svc_req *req )
{
        static RE_get_current_template_result argzz;
        static char     template_buff[MAX_TEMPLATE_LEN] = "";
```



```
        setLastRpcTime( );      /* note time of last RPC */

        /* init output struct ptr first time; clear string other times */
        if (template_buff[0] == 0)
                argzz.templateName = template_buff;
        else
                template_buff[0] = 0;

        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else {
                argzz.status = RSTSL_GetCurrentTemplate( argzz.templateName,
                                                         &argzz.alternate );
        }

        set_rpc_obj( re_get_current_template, &argzz.RPCobjID );

        return &argzz;
}

/****************************************************************************
 **
 ** Routine:  re_get_necessary_media
 **
 ** Inputs:  RE_get_necessary_media_args  * - args for the RPC call
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **          RE_get_necessary_media_result * - result of RPC function call
 **
 ** Purpose:  Function to retrieve the list of media need to restore the
 **       currently marked objects
 **
 ** Intended caller:  Internal Only.
 ****************************************************************************
 */

RE_get_necessary_media_result *
re_get_necessary_media_1_svc(IN RE_get_necessary_media_args *arg,
                             IN struct svc_req *req )
{
        static RE_get_necessary_media_result argzz;
        static RSTRPC_media_list        *media_list = NULL;

        setLastRpcTime( );      /* note time of last RPC */

        /* free previosly returned list of media */
        if (media_list) {
                RSTSL_FreeMediaObjectList( media_list );
                media_list = NULL;
        }

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        else if ( (argzz.status = check_RPC_state(
                                                FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else {
                /* init result structure */
                argzz.numEntries = 0;
                argzz.cookie = arg->cookie;
```

```
                argzz.mediaList = NULL;

                argzz.status = RSTSL_GetNecessaryMedia( arg->maxEntries,
                                                        &argzz.mediaList,
                                                        &argzz.numEntries,
                                                        arg->all,
                                                        &argzz.cookie );


                media_list = argzz.mediaList;
                                                /* save to free next time in */
        }


    set_rpc_obj( re_get_necessary_media, &argzz.RPCobjID );

    return &argzz;
}

/*****************************************************************************
**
** Routine:  re_get_all_backup_times
**
** Inputs:  RE_get_all_backup_times_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_status_result * - result of RPC function call
**
** Purpose:  Function to start the asynchronous operation to find all the
**           backups available for the current workitem
**
** Intended caller:  RPC call from Restore API client
*****************************************************************************
*/

RE_status_result *
re_get_all_backup_times_1_svc(IN RE_get_all_backup_times_args *arg,
                              IN struct svc_req *req )
{
        static RE_status_result                 argzz;
        RE_get_all_backup_times_args    *cmd_args;
        int                             status;
        setLastRpcTime( );      /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        cmd_args = calloc( 1, sizeof(RE_get_all_backup_times_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                                "Cannot malloc RE_get_all_backup_times_args"
                                                                        );

                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no RPC is in progress */
        else if (E_SUCCESS != (argzz.status = check_RPC_state( TRUE,
                                        COMMAND_GET_ALL_TIMES )) )
                ;       /* just return failure status */
        else {
                cmd_args->startTime = arg->startTime;
                cmd_args->endTime = arg->endTime;
                cmd_args->flags = arg->flags;
                cmd_args->maxEntries = arg->maxEntries;
```



```
                cmd_args->cookie = arg->cookie;

                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        status, 0,
                                                "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand( COMMAND_GET_ALL_TIMES, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        status, 0,
                                                "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                        argzz.status = E_SUCCESS;
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_get_all_backup_times_args,
                                  (char *)cmd_args);
                        free( cmd_args );
                }
        }

        set_rpc_obj( re_get_all_backup_times, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_get_all_backup_times_result
**
** Inputs:  RE_null_args  * - No args for this RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_get_all_backup_times_result * - result of RPC function call
**
** Purpose:  Function to test for completion of the re_get_all_backup_times
**           RPC call, and retrieve some or all of its output.
**
** Intended caller:  RPC call from Restore API client
*****************************************************************************
*/

RE_get_all_backup_times_result *
re_get_all_backup_times_result_1_svc( IN RE_null_args *arg,
                                      IN struct svc_req *req )
{

        static RE_get_all_backup_times_result argzz;
        static RE_get_all_backup_times_result *outerg = NULL;
        int     result, cmd, status;
```

```
        setLastRpcTime( );      /* note time of last RPC */

        if (outarg)
        {       /* free last results */
                outarg->backupTimes = NULL;  /* this is freed by RSTSL_... */
                xdr_free( xdr_RE_get_all_backup_times_result,
                          (char *)outarg );
                free( outarg );
                outarg = NULL;
        }

        else
        {       /* init static output struct for errors (
                                                    1st time & aft errs */
                argzz.numEntries = 0;
                argzz.cookie = 0;
                argzz.backupTimes = NULL;
        }

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        /* make sure this RPC is in progress */
        if (E_SUCCESS != (argzz.status = check_RPC_state( FALSE,
                                        COMMAND_GET_ALL_TIMES )) )
                ;       /* just return failure status */

        /* test for completion of processing */
        else if (PopResult( -1, &result, &cmd, &status) )
        {
                if (status == COMMAND_RECORD_GET_FAILED)
                {
                        argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                } else {
                        /* log error, clean up, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                        status, 0, "PopResult failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
        }
        else if (cmd != COMMAND_GET_ALL_TIMES)
        {
                /* log error, clean up, return error  */
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                MESSAGE_INVALID_COMMAND, 0,
                        "PopResult mismatch: got %d command, expected %d\n",
                                cmd, COMMAND_GET_ALL_TIMES);
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (result != COMMAND_RESULT_SUCCESS)
        {
               EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                MESSAGE_FAILURE_DOING_ASYNC_RPC , 0,
                                "RPC failure in process manager thread" );
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (PopRpcOutput( (void **)&outarg, &status) )
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status,
                                0, "PopRpcOutput failure");
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else
        {       /* return popped results struct */
            set_rpc_obj( re_get_all_backup_times_result, &outarg->RPCobjID);
            clear_RPC_state( );
```



```
                return outarg;                  /* indicate process mgr idle */
        }

        /* return static result struct on errors */
        set_rpc_obj( re_get_all_backup_times_result, &argzz.RPCobjID );
        if (argzz.status == EP_RB_RECOVER_SERVERFAIL)
            clear_RPC_state( );
                                        /* indicate process mgr idle on fatals */
        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_get_current_backup_time
**
** Inputs:  RE_null_args  * - args for the RPC call (none)
**
** Outputs:  None
**
** Return Codes:
**       RE_get_current_backup_time_result * - result of RPC function call
**
** Purpose:  Function to retrieve the currently selected backup time
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_get_current_backup_time_result *
re_get_current_backup_time_1_svc(
                                IN RE_null_args *arg, IN struct svc_req *req )
{
        static RE_get_current_backup_time_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        /* init result structure */
        argzz.backupTime = 0;

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;       /* we weren't idle, reject call */
        else {
                argzz.status = RSTSL_GetCurrentBackupTime(
                                                &argzz.backupTime );
        }

        set_rpc_obj( re_get_current_backup_time, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_is_there_prev_backup
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
```

```
**          RE_boolean_result * - result of RPC function call
**
** Purpose:  Function to test if there is an older backup available
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_boolean_result *
re_is_there_prev_backup_1_svc( IN RE_set_backup_time_args *arg,
                               IN struct svc_req *req )
{
        static RE_boolean_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else {
                argzz.status = RSTSL_IsTherePrevBackup( arg->flags,
                                                        &argzz.boolResult );
        }

        set_rpc_obj( re_is_there_prev_backup, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_is_there_next_backup_for_time
**
** Inputs:  RE_backup_for_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_boolean_result * - result of RPC function call
**
** Purpose:  Function to test if there is a backup available more recent than a
**       specified time.
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_boolean_result *
re_is_there_next_backup_for_time_1_svc(IN RE_backup_for_time_args *arg,
                               IN struct svc_req *req )
{
        static RE_boolean_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
```



```
                ;       /* we weren't idle, reject call */
        else {
                argzz.status = RSTSL_IsThereNextBackupForTime( arg->time,
                                                               arg->flags,
                                                               &argzz.
                                                                 boolResult );
        }

        set_rpc_obj( re_is_there_next_backup_for_time, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_is_there_next_backup
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_boolean_result * - result of RPC function call
**
** Purpose:  Function to test if there is a backup available more recent than
**       the currently selected time.
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_boolean_result *
re_is_there_next_backup_1_svc(IN RE_set_backup_time_args *arg,
                               IN struct svc_req *req )
{
        static RE_boolean_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else {
                argzz.status = RSTSL_IsThereNextBackup( arg->flags,
                                                        &argzz.boolResult );
        }

        set_rpc_obj( re_is_there_next_backup, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  set_backup_time_request
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**          int         internal_command
**          int rpc_function_no
**
** Outputs:  None
**
```

```
** Return Codes:
**          RE_status_result * - result of RPC function call
**
** Purpose:  Function to start the asynchronous operation of all the
**         re_set..._backup rpc functions
**
** Intended caller:  RPC function service calls
*****************************************************************************
*/

RE_status_result *
set_backup_time_request(IN RE_set_backup_time_args *arg,
                        IN int internal_command,
                        IN int rpc_function_no )
{
        static RE_status_result                 argzz;
        RE_set_backup_time_args                 *cmd_args;
        int                                     status;
        setLastRpcTime( );      /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        cmd_args = calloc( 1, sizeof(RE_set_backup_time_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                                "Cannot malloc RE_set_backup_time_args" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no RPC is in progress */
        else if (E_SUCCESS != (argzz.status = check_RPC_state( TRUE,
                                        internal_command )) )
                ;       /* just return failure status */
        else {
                cmd_args->flags = arg->flags;

                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                              "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand( internal_command, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                              "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                        argzz.status = E_SUCCESS;
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_set_backup_time_args,
                                  (char *)cmd_args);
                        free( cmd_args );
                }
        }

        set_rpc_obj( rpc_function_no, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  set_backup_time_result
**
** Inputs:  int  internal_command
**          int rpc_function_no
**
** Outputs:  None
**
** Return Codes:
**          RE_status_result * - result of RPC function call
**
** Purpose:  Function to test for completion of the re_set_xxx_backup
**           RPC calls, and retrieve some or all of their output.
**
** Intended caller:  RPC service function
*****************************************************************************
*/

RE_status_result *
set_backup_time_result( IN int internal_command, IN int rpc_function_no )
{

        static RE_status_result argzz;
        static RE_status_result *outarg = NULL;
        int     result, cmd, status;

        setLastRpcTime( );      /* note time of last RPC */

        if (outarg)
        {       /* free last results */
                xdr_free( xdr_RE_status_result, (char *)outarg );
                free( outarg );
                outarg = NULL;
        }

        /* make sure this RPC is in progress */
        if (E_SUCCESS != (argzz.status = check_RPC_state( FALSE,
                                        internal_command )) )
                ;       /* just return failure status */

        /* test for completion of processing */
        else if (PopResult( -1, &result, &cmd, &status) )
        {
                if (status == COMMAND_RECORD_GET_FAILED)
                {
                        argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                } else {
                        /* log error, clean up, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0, "PopResult failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
        }
        else if (cmd != internal_command)
        {
                /* log error, clean up, return error  */
```

```
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_INVALID_COMMAND, 0,
                        "PopResult mismatch: got %d command, expected %d\n",
                                      cmd, internal_command);
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (result != COMMAND_RESULT_SUCCESS)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_FAILURE_DOING_ASYNC_RPC , 0,
                                      "RPC failure in process manager thread" );
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (PopRpcOutput( (void **)&outarg, &status) )
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status,
                                      0, "PopRpcOutput failure");
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else
        {
                /* return popped results struct */
            set_rpc_obj( rpc_function_no, &outarg->RPCobjID);
            clear_RPC_state( );
                                        /* indicate process mgr idle */
            return outarg;
        }

        /* return static result struct on errors */
        set_rpc_obj( rpc_function_no, &argzz.RPCobjID );
        if (argzz.status == EP_RB_RECOVER_SERVERFAIL)
            clear_RPC_state( );
                                /* indicate process mgr idle on fatals */
        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_set_first_backup
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_status_result * - result of RPC function call
**
** Purpose:  Function to select the oldest backup for the current workitem
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_set_first_backup_1_svc(
                IN RE_set_backup_time_args *arg, IN struct svc_req *req )
{
        RE_status_result *argzz;

        argzz = set_backup_time_request( arg,
                                         COMMAND_SET_FIRST_BACKUP,
                                         re_set_first_backup );

        return argzz;
}

/*****************************************************************************
```



```
**
** Routine:  re_set_first_backup_result
**
** Inputs:  RE_null_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_status_result * - result of RPC function call
**
** Purpose:  Function to test for completion of the rpc to select the oldest
**       backup for the current workitem
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_set_first_backup_result_1_svc( IN RE_null_args *arg,
                                  IN struct svc_req *req )
{
        RE_status_result *argzz;

        argzz = set_backup_time_result( COMMAND_SET_FIRST_BACKUP,
                                        re_set_first_backup );

        return argzz;
}


/*****************************************************************************
**
** Routine:  re_set_next_backup
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**          RE_status_result * - result of RPC function call
**
** Purpose:  Function to set to the next (more recent) backup time
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_set_next_backup_1_svc(
                IN RE_set_backup_time_args *arg, IN struct svc_req *req )
{
        RE_status_result *argzz;

        argzz = set_backup_time_request( arg,
                                         COMMAND_SET_NEXT_BACKUP,
                                         re_set_next_backup );

        return argzz;
}
/*****************************************************************************
**
** Routine:  re_set_next_backup_result
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
```

```
 **
 ** Return Codes:
 **           RE_status_result * - result of RPC function call
 **
 ** Purpose:  Function to set to the next (more recent) backup time
 **
 ** Intended caller:  Internal Only.
 ********************************************************************************
 */

RE_status_result *
re_set_next_backup_result_1_svc(
                              IN RE_null_args *arg, IN struct svc_req *req )
{
        RE_status_result *argzz;

        argzz = set_backup_time_result( COMMAND_SET_NEXT_BACKUP,
                                        re_set_next_backup );

        return argzz;
}


/*******************************************************************************
**
** Routine:  re_set_prev_backup
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_status_result * - result of RPC function call
**
** Purpose:  Function to set to the next (more recent) backup time
**
** Intended caller:  Internal Only.
********************************************************************************
*/

RE_status_result *
re_set_prev_backup_1_svc(
                    IN RE_set_backup_time_args *arg, IN struct svc_req *req )
{
        RE_status_result *argzz;

        argzz = set_backup_time_request( arg,
                                         COMMAND_SET_PREVIOUS_BACKUP,
                                         re_set_prev_backup );

        return argzz;
}
/*******************************************************************************
**
** Routine:  re_set_prev_backup_result
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_status_result * - result of RPC function call
**
** Purpose:  Function to set to the next (more recent) backup time
**
** Intended caller:  Internal Only.
```



```
 ********************************************************************************
 */

RE_status_result *
re_set_previous_backup_result_1_svc(
                              IN RE_null_args *arg, IN struct svc_req *req )
{
        RE_status_result *argzz;

        argzz = set_backup_time_result( COMMAND_SET_PREVIOUS_BACKUP,
                                        re_set_prev_backup );

        return argzz;
}

/*******************************************************************************
**
** Routine:  re_set_backup_for_time
**
** Inputs:  RE_backup_for_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_status_result * - result of RPC function call
**
** Purpose:  Function to set to a specified backup time.
**
** Intended caller:  Internal Only.
********************************************************************************
*/

RE_status_result *
re_set_backup_for_time_1_svc( IN RE_backup_for_time_args *arg,
                              IN struct svc_req *req )
{


        static RE_status_result                 argzz;
        RE_backup_for_time_args                 *cmd_args;
        int                                     status;
        setLastRpcTime( );      /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        cmd_args = calloc( 1, sizeof(RE_backup_for_time_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                                "Cannot malloc RE_get_all_backup_times_args"
                                                                        );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no RPC is in progress */
        else if (E_SUCCESS != (argzz.status = check_RPC_state( TRUE,
                                        COMMAND_SET_BACKUP_FOR_TIME )) )
                ;       /* just return failure status */
        else {
                cmd_args->flags = arg->flags;
                cmd_args->time = arg->time;

                if (PushRpcInput( (void *)cmd_args, &status) )
```

```
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                  "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand( COMMAND_SET_BACKUP_FOR_TIME, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                  "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                        argzz.status = E_SUCCESS;
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_backup_for_time_args,
                                  (char *)cmd_args);
                        free( cmd_args );
                }
        }

        set_rpc_obj( re_set_backup_for_time, &argzz.RPCobjID );

        return &argzz;
}
/*****************************************************************************
**
** Routine:  re_set_backup_for_time_result
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_status_result * - result of RPC function call
**
** Purpose:  Function to set to the next (more recent) backup time
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_set_backup_for_time_result_1_svc(
                        IN RE_null_args *arg, IN struct svc_req *req )
{
        RE_status_result *argzz;

        argzz = set_backup_time_result( COMMAND_SET_BACKUP_FOR_TIME,
                                        re_set_backup_for_time );

        return argzz;
}

/*****************************************************************************
**
```



```
** Routine:  re_is_there_prev_backup_for_time_1
**
** Inputs:   RE_backup_for_time_args * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_boolean_result * - result of RPC function call
**
** Purpose:  Function to determine if there is an older backup available.
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_boolean_result *
re_is_there_prev_backup_for_time_1_svc(IN RE_backup_for_time_args *arg,
                                       IN struct svc_req *req )
{
        static RE_boolean_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else {
                argzz.status = RSTSL_IsTherePrevBackupForTime( arg->time,
                                                               arg->flags,
                                                        &argzz.
                                                            boolResult );
        }

        set_rpc_obj( re_is_there_prev_backup_for_time, &argzz.RPCobjID );

        return &argzz;
}



/*****************************************************************************
**
** Routine:  re_set_most_recent_backup
**
** Inputs:  RE_set_backup_time_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_status_result * - result of RPC function call
**
** Purpose:  Function to set to the next (more recent) backup time
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_set_most_recent_backup_1_svc(
                        IN RE_set_backup_time_args *arg, IN struct svc_req *req )
```

```
{
        RE_status_result *argzz;

        argzz = set_backup_time_request( arg,
                                         COMMAND_SET_MOST_RECENT_BACKUP,
                                         re_set_prev_backup );

        return argzz;
}
/*****************************************************************************
 **
 ** Routine:  re_set_most_recent_backup_result
 **
 ** Inputs:  RE_set_backup_time_args  * - args for the RPC call
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **           RE_status_result * - result of RPC function call
 **
 ** Purpose:  Function to set to the next (more recent) backup time
 **
 ** Intended caller:  Internal Only.
 *****************************************************************************
 */

RE_status_result *
re_set_most_recent_backup_result_1_svc(
                                IN RE_null_args *arg, IN struct svc_req *req )
{
        RE_status_result *argzz;

        argzz = set_backup_time_result( COMMAND_SET_MOST_RECENT_BACKUP,
                                        re_set_most_recent_backup );

        return argzz;
}

/*****************************************************************************
 **
 ** Routine:  re_get_host_platform_type_1
 **
 ** Inputs:   RE_string_args * - args for the RPC call
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **           RE_get_host_platform_type_result * - result of RPC function call
 **
 ** Purpose:  Function to retrieve the platform type of the specified host
 **
 ** Intended caller:  Internal Only.
 *****************************************************************************
 */

RE_get_host_platform_type_result *
re_get_host_platform_type_1_svc(
                               IN RE_string_args *arg, IN struct svc_req *req )
{
        static RE_get_host_platform_type_result argzz;

        setLastRpcTime( );       /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;
```



```
        else if ( (argzz.status = check_RPC_state(
                                                FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)          /* if not idle, trouble */
                ;        /* we weren't idle, reject call */
        else {
                argzz.status = RSTSL_GetHostPlatformType( arg->name,
                                                          &argzz.ptype );
        }

        set_rpc_obj( re_get_host_platform_type, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
 **
 ** Routine:  re_does_alternate_exist
 **
 ** Inputs:  RE_does_alternate_exist_args  * - args for the RPC call
 **
 ** Outputs:  None
 **
 ** Return Codes:
 **           RE_boolean_result * - result of RPC function call
 **
 ** Purpose:  Function to test if there is an alternate backup trailset
 **       available for the specified template
 **
 ** Intended caller:  Internal Only.
 *****************************************************************************
 */

RE_boolean_result *
re_does_alternate_exist_1_svc( IN RE_does_alternate_exist_args *arg,
                               IN struct svc_req *req )
{
        static RE_boolean_result argzz;

        setLastRpcTime( );       /* note time of last RPC */

        if (NULL == arg)
                argzz.status = EP_RB_RECOVER_RPC_FAIL;

        else if ( (argzz.status = check_RPC_state(
                                                FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)          /* if not idle, trouble */
                ;        /* we weren't idle, reject call */
        else {
                argzz.status = RSTSL_DoesAlternateExist( arg->topLevelObj,
                                                         arg->templateName,
                                                         &argzz.boolResult );
        }

        set_rpc_obj( re_does_alternate_exist, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
 **
 ** Routine:  re_finish_1
 **
 ** Inputs:   RE_null_args * - args for the RPC call (none)
 **
 ** Outputs:  None
 **
```

```
** Return Codes:
**         RE_status_result * - result of RPC function call
**
** Purpose:  Function to terminate the restore session at the browse stage
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_finish_1_svc(IN RE_null_args *arg, IN struct svc_req *req )
{
        static RE_status_result argzz;
        RE_null_args    *cmd_args;
        int     status;

        setLastRpcTime( );      /* note time of last RPC */

        cmd_args = calloc( 1, sizeof(RE_null_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                                      "Cannot malloc RE_null_args" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        else if ( (argzz.status = check_RPC_state(
                                        TRUE, COMMAND_NONE_ACTIVE ) )
                != E_SUCCESS)   /* if idle, stay idle */
                ;       /* we weren't idle, reject finish */
        else
        {
                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                              "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else if (PushCommand( COMMAND_FINISH, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                              "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else
                        argzz.status = E_SUCCESS;
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_null_args, (char *)cmd_args );
                        free( cmd_args );
                }
        }

        set_rpc_obj( re_finish, &argzz.RPCobjID );

        return &argzz;
}
```



```
/*****************************************************************************
**
** Routine:  re_ping_1
**
** Inputs:   RE_null_args * - args for the RPC call (none)
**
** Outputs:  None
**
** Return Codes:
**         RE_status_result * - result of RPC function call
**
** Purpose:  Function to keep the engine alive
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_ping_1_svc(IN RE_null_args *arg, IN struct svc_req *req )
{
  static RE_status_result argzz;

  setLastRpcTime( );      /* note time of last RPC */

  argzz.status = E_SUCCESS;

  return &argzz;
}



/*****************************************************************************
**
** Routine:  re_get_marked_total_size
**
** Inputs:  RE_null_args  * - args for the RPC call (none)
**
** Outputs:  None
**
** Return Codes:
**         RE_get_marked_total_size_result * - result of RPC function call
**
** Purpose:  Function to return the total size of the objects currently marked
**       for restore
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_get_marked_total_size_result *
re_get_marked_total_size_1_svc(IN RE_null_args *arg, IN struct svc_req *req )
{
        static RE_get_marked_total_size_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        argzz.total.high = 0;
        argzz.total.low = 0;

        if ( (argzz.status = check_RPC_state( FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else
        {
```

```
                argzz.total = RSTSL_GetMarkedTotalSize( );
                argzz.status = E_SUCCESS;
        }

    set_rpc_obj( re_get_marked_total_size, &argzz.RPCobjID );

    return &argzz;
}

/*****************************************************************************
**
** Routine:  re_is_object_marked_1
**
** Inputs:   RE_is_object_marked_args * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_is_object_marked_result * - result of RPC function call
**
** Purpose:  Function to determine if specified object is marked for restore
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_is_object_marked_result *
re_is_object_marked_1_svc(
                IN RE_is_object_marked_args *arg, IN struct svc_req *req )
{
        static RE_is_object_marked_result argzz;
        static marked_len = 0;

        setLastRpcTime( );      /* note time of last RPC */

        /* free previosly calloc's bool array */
        if (marked_len) {
                free( argzz.marked.marked_val );
                marked_len = 0;
        }

        /* init result structure */
        argzz.numMarked = 0;
        argzz.marked.marked_len = 0;
        argzz.marked.marked_val = NULL;

        if (NULL == arg || NULL == arg->objList || arg->numEntries <= 0)
                argzz.status = EP_RB_RECOVER_BAD_ARGS;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else if (NULL == (argzz.marked.marked_val =
                        calloc( arg->numEntries, sizeof(bool_t) )) )
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                                      "Cannot malloc bool_t array" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        else {
                argzz.marked.marked_len = marked_len = arg->numEntries;
                argzz.status = RSTSL_IsObjectMarked( arg->numEntries,
                                                     arg->objList,
                                                     &argzz.numMarked,
```



```
                                                     argzz.marked.
                                                         marked_val );
        }

        set_rpc_obj( re_is_object_marked, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_is_object_markable
**
** Inputs:  RE_is_object_markable_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_is_object_markable_result * - result of RPC function call
**
** Purpose:  Function to test if the specified object is markable
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_is_object_markable_result *
re_is_object_markable_1_svc(IN RE_is_object_markable_args *arg,
                        IN struct svc_req *req )
{
        static RE_is_object_markable_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        argzz.markable = FALSE;
        if (NULL == arg || NULL == arg->thisObject)
                argzz.status = EP_RB_RECOVER_BAD_ARGS;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else
        {
                argzz.markable = RSTSL_IsObjectMarkable( arg->thisObject );
                argzz.status = E_SUCCESS;
        }

        set_rpc_obj( re_is_object_markable, &argzz.RPCobjID );

        return &argzz;
}

/*****************************************************************************
**
** Routine:  re_find_restorable_objects_1
**
** Inputs:   RE_find_restorable_objects_args * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_find_restorable_objects_result * - result of RPC function call
**
** Purpose:  Function to search for restorable objects in the backup catalog
**
```

```
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_find_restorable_objects_result *
re_find_restorable_objects_1_svc( IN RE_find_restorable_objects_args *arg,
                                  IN struct svc_req *req )
{
        static RE_find_restorable_objects_result argzz;
        RE_find_restorable_objects_args                 *cmd_args;
        int                                             status;

        setLastRpcTime( );      /* note time of last RPC */

        cmd_args = calloc( 1, sizeof (RE_find_restorable_objects_args) );
        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_NO_MEMORY, errno,
                                "Cannot malloc RE_find_restorable_objects_args"
                                                                        );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no rpc is in progress */
        else if ( (argzz.status = check_RPC_state( TRUE,
                                        COMMAND_FIND_RESTORABLE_OBJECTS ))
          != E_SUCCESS )
                ;       /* just return failure status */

        else
        {
                ClearRpcCancelFlag( );  /* reset cancel flag */
                ClearProgressValue( );  /* reset progress count */

                cmd_args->searchCriteria = arg->searchCriteria;
                arg->searchCriteria = NULL;     /* to avoid 2 frees */

                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand(
                                COMMAND_FIND_RESTORABLE_OBJECTS, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                        argzz.status = E_SUCCESS;
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_find_restorable_objects_args,
                                  (char *)cmd_args );
```



```
                        free( cmd_args );
                }
        }

    set_rpc_obj( re_find_restorable_objects, &argzz.RPCobjID );

    return &argzz;
}

/*****************************************************************************
**
** Routine:  re_get_find_results
**
** Inputs:  RE_get_find_results_args  * - args for the RPC call
**
** Outputs:  None
**
** Return Codes:
**           RE_get_find_results_result * - result of RPC function call
**
** Purpose:  Function to retrieve the results of the find_restorable objects
**       request
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_get_find_results_result *
re_get_find_results_1_svc(
                        IN RE_get_find_results_args *arg, IN struct svc_req *req )
{
        static RE_get_find_results_result       argzz;
        RE_find_restorable_objects_result       *outarg = NULL;
        static RSTRPC_found_obj_list            *last_list = NULL;
        int     result, cmd, status;

        setLastRpcTime( );      /* note time of last RPC */

        if (last_list)
        {       /* free last results */
                xdr_free( xdr_RSTRPC_found_obj_list, (char *)last_list );
                last_list = NULL;
        }

        /* init static output struct */
        argzz.numEntries = 0;
        argzz.cookie = arg->cookie;
        argzz.foundObjs = NULL;

        /* If interrupt was requested, make sure find was running */
        if (arg->interrupt)
        {
                if (E_SUCCESS != (argzz.status =
                    check_RPC_state(
                                        FALSE, COMMAND_FIND_RESTORABLE_OBJECTS )))
                {
                /* for get find results after first good get results call: */
                /* find not active -- make sure idle */
                    argzz.status = check_RPC_state (
                                        FALSE, COMMAND_NONE_ACTIVE);
                /* status = E_SUCCESS means call only GetFindResults */
                }

                /* test for completion of find processing: */
                else if (PopResult( 1, &result, &cmd, &status) )
                {
```

```
                    if (status == COMMAND_RECORD_GET_FAILED)
                    {
                        /* still going: signal cancel, wait till done */
                        SetRpcCancelFlag( );
                        if (PopResult( MAX_CANCEL_WAIT_SECS, &result,
                                       &cmd, &status) )
                            /* if no result, error */
                            argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        else        /* indicate canceled anyway */
                            argzz.status = EP_RB_RECOVER_FIND_INTERRUPTED;
                    }
                    else {
                        /* log pop error, clean up, return error  */
                        EDMRestoreEng_logent(
                                                __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                    "PopResult failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                    }
                }
                else    /* indicate canceled anyway */
                    argzz.status = EP_RB_RECOVER_FIND_INTERRUPTED;
            }
            else    /* didn't interrupt -- see if still running: */
            {
                if (E_SUCCESS == (argzz.status =
                    check_RPC_state(
                                FALSE, COMMAND_FIND_RESTORABLE_OBJECTS )))
                {
                    /* was doing find, test for completion of processing: */
                    if (PopResult( 1, &result, &cmd, &status) )
                    {
                        if (status == COMMAND_RECORD_GET_FAILED)
                        {   /* not done yet */
                            argzz.numEntries = ReadProgressValue( );
                            argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                        }
                        else
                        {   /* log error, clean up, return error  */
                            EDMRestoreEng_logent(
                                                __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                    "PopResult failed");
                            argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        }
                    }
                    else        /* pop worked: indicate results need popping
                                                                        */
                        argzz.status = EP_RB_RECOVER_FIND_INTERRUPTED;
                }
                /* for get find results after first get find results call */
                else if (E_SUCCESS != (argzz.status =
                        check_RPC_state( FALSE, COMMAND_NONE_ACTIVE )))
                    ;       /* another cmd running, invalid */
                /* argzz.status = E_SUCCESS means call only GetFindResults */
            }

            if (EP_RB_RECOVER_FIND_INTERRUPTED == argzz.status)
            {
            /* popped result, validate and pop output: */
                if (cmd != COMMAND_FIND_RESTORABLE_OBJECTS)
                {
                    /* log error, clean up, return error  */
                    EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                          MESSAGE_INVALID_COMMAND, 0,
                                "PopResult mismatch: got %d command,
```



```
                                                        expected %d\n",
                                        cmd,
                                          COMMAND_FIND_RESTORABLE_OBJECTS);
                    argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else if (result != COMMAND_RESULT_SUCCESS)
                {
                    EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                          MESSAGE_FAILURE_DOING_ASYNC_RPC, 0,
                                    "RPC failure in process manager thread"
                                                                        );
                    argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else if (PopRpcOutput( (void **)&outarg, &status) || (
                                                                outarg) )
                {
                    EDMRestoreEng_logent(
                                        __FILE__, __LINE__, LOG_ERR, status,
                                          0, "PopRpcOutput failure");
                    argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
                else
                {
                        argzz.status = outarg->status;  /* get real status */
                        /* free results */
                        xdr_free( xdr_RE_find_restorable_objects_result,
                                  (char *)outarg );
                        free( outarg );
                        outarg = NULL;
                }

                clear_RPC_state( );
                                                /* indicate process mgr idle */
            }

            if ( E_SUCCESS == argzz.status
              || EP_RB_RECOVER_FIND_INTERRUPTED == argzz.status )
            {
                    /* canceled or done, get some data or free it */
                    argzz.status = RSTSL_GetFindResults( arg->interrupt,
                                                         arg->maxEntries,
                                                         &argzz.foundObjs,
                                                         &argzz.numEntries,
                                                         &argzz.cookie );
                    last_list = argzz.foundObjs;
            }

            /* return static results struct */
            set_rpc_obj( re_get_find_results, &argzz.RPCobjID );

            if (argzz.status == EP_RB_RECOVER_SERVERFAIL)
                clear_RPC_state( );
                                        /* indicate process mgr idle on fatals */
            return &argzz;
        }
        /* end of:  re_get_find_results */

/*****************************************************************************
**
** Routine:  re_is_object_searchable
**
** Inputs:  RE_tlo_query_args  * - args for the RPC call
**
** Outputs:  none
**
```

```
** Return Codes:
**          RE_boolean_result *
**
** Purpose:  Function to test if the specified object supports the find api
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_boolean_result *
re_is_object_searchable_1_svc(IN RE_tlo_query_args *arg,
                              IN struct svc_req *req )

{
        static RE_boolean_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        argzz.boolResult = FALSE;
        if (NULL == arg || NULL == arg->topLevelObj)
                argzz.status = EP_RB_RECOVER_BAD_ARGS;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else
        {
                argzz.boolResult = RSTSL_IsObjectSearchable(
                                                        arg->topLevelObj );
                argzz.status = E_SUCCESS;
        }

        set_rpc_obj( re_is_object_searchable, &argzz.RPCobjID );

        return &argzz;
}


/*****************************************************************************
**
** Routine:  re_get_backup_times_support
**
** Inputs:  RE_tlo_query_args  * - args for the RPC call
**
** Outputs:  none
**
** Return Codes:
**          RE_boolean_result *
**
** Purpose:  Function to test if the specified object supports restores from
**           multiple backup times
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_boolean_result *
re_get_backup_times_support_1_svc( IN RE_tlo_query_args *arg,
                                   IN struct svc_req *req )
{
        static RE_boolean_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        argzz.boolResult = FALSE;
        if (NULL == arg || NULL == arg->topLevelObj)
```



```
                argzz.status = EP_RB_RECOVER_BAD_ARGS;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else
        {
                argzz.boolResult = RSTSL_GetBackupTimesSupport
                                                        (arg->topLevelObj);
                argzz.status = E_SUCCESS;
        }

        set_rpc_obj( re_get_backup_times_support, &argzz.RPCobjID );

        return &argzz;
}


/*****************************************************************************
**
** Routine:  re_get_symm_restore_option
**
** Inputs:  RE_tlo_query_args  * - args for the RPC call
**
** Outputs:  none
**
** Return Codes:
**          RE_boolean_result *
**
** Purpose:  Function to test if the specified object supports restores
**           through the Symm
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_boolean_result *
re_get_symm_restore_option_1_svc( IN RE_tlo_query_args *arg,
                                  IN struct svc_req *req )

{
        static RE_boolean_result argzz;

        setLastRpcTime( );      /* note time of last RPC */

        argzz.boolResult = FALSE;
        if (NULL == arg || NULL == arg->topLevelObj)
                argzz.status = EP_RB_RECOVER_BAD_ARGS;

        else if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_NONE_ACTIVE ))
          != E_SUCCESS)         /* if not idle, trouble */
                ;               /* we weren't idle, reject call */
        else
        {
                argzz.boolResult = RSTSL_GetSymmRestoreOption
                                                        (arg->topLevelObj);
                argzz.status = E_SUCCESS;
        }

        set_rpc_obj( re_get_symm_restore_option, &argzz.RPCobjID );

        return &argzz;
}
```

```
/*****************************************************************************
**
** Routine:  set_rpc_obj
**
** Inputs:   rpc_id       rpc function number
**           rpc_objID  pointer to RPC object ID
**
** Outputs:  None
**
** Return Codes:
**           none
**
** Purpose:  load rpc object id with rpc number and timestamp
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

static void set_rpc_obj( ulong rpc_id, RE_rpc_objID *rpc_objID )
{
        struct timeval timeofday;
        void *dummy = NULL;

        rpc_objID->rpc_type = rpc_id;
        gettimeofday( &timeofday, dummy );
        rpc_objID->time = timeofday.tv_sec;
        return;
}

/*****************************************************************************
**
** Routine:  check_RPC_state
**
**       Function to check if there is any current command, or if it is set to
**       a specific value, and optionally, to set it to a new command value
**
** Inputs:   bool set -          indicates whether this is a request to set
                                                                        the
**                              current command (1/true), or just to check it
**
**           int cmd -          if set input is 0/false,
                                                        command value to check
**                              for (COMMAND_NONE_ACTIVE means idle)
**                              if set is 1/true, value to change current
**                              command to,
                                        after verifying that is it not set,
**                              i.e., that it is set to COMMAND_NONE_ACTIVE.
**
** Outputs:  None
**
** Return Codes:
**           RE_errno_ty result  - result of check: E_SUCCESS if current
**                                 command was in desired state;
**                                 EP_RB_RECOVER_INVALOP otherwise
**
** Purpose:  verify that no async RPC is actice,
                                                or that specified one IS active
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

static RE_errno_ty check_RPC_state( boolean_ty set, int cmd )
{
        if ( (!set && cmd != current_rpc_cmd)
           || (set && current_rpc_cmd != COMMAND_NONE_ACTIVE))
```



```
                /* check-only failure or can't set because another RPC busy
                                                                        */
                return EP_RB_RECOVER_INVALOP;
        else {  /* check succeeded, change current cmd if requested */
                if (set)
                        current_rpc_cmd = cmd;
                return E_SUCCESS;
        }
}

/*****************************************************************************
**
** Routine:  clear_RPC_state
**
**       Function to clear the current RPC command
**
** Inputs:   none
**
** Outputs:  None
**
** Return Codes:          none
**
** Purpose:  indicate that no async RPC is active
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

static void clear_RPC_state(
                        void )              /* indicate process mgr idle */
{
        current_rpc_cmd = COMMAND_NONE_ACTIVE;
}


/*****************************************************************************
**
** Routine: re_load_recx_directives
**
** Inputs:  RE_recx_file_info  * - args for the RPC call to get directives
                                                                        file
**
** Outputs: RE_status_result * - result of RPC function call
**
** Purpose:  Function to load the recx file into the recx struct and then
**           into context structure
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_load_recx_directives_1_svc( IN RE_recx_file_info *arg,
                               IN struct svc_req *req )
{
        static RE_status_result          argzz;
       RSTRPC_recx_file_info       *fileinfo;
        RSTRPC_recx_file_info       *cmd_args;
        int                          status;
        cmd_args = calloc( 1, sizeof( RSTRPC_recx_file_info) );

        fileinfo = &arg->fileinfo;

        if (NULL == cmd_args)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
```

```
                                        MESSAGE_NO_MEMORY, errno,
                        "Cannot malloc RE_recx_file_info structure" );
                argzz.status = EP_RB_RECOVER_NOMEM;
        }
        /* make sure no rpc is in progress */
        else if ( (argzz.status = check_RPC_state( TRUE,
                 COMMAND_LOAD_RECX_DIRECTIVES )) != E_SUCCESS )
                ;       /* just return failure status */

        else
        {
                ClearRpcCancelFlag( );  /* reset cancel flag */
                ClearProgressValue( );  /* reset progress count */

                cmd_args->hostname = esl_strdup( fileinfo->hostname );
                cmd_args->filename = esl_strdup( fileinfo->filename );

                if (PushRpcInput( (void *)cmd_args, &status) )
                {
                        /* log error, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                "PushRpcInput failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else if (PushCommand(
                                        COMMAND_LOAD_RECX_DIRECTIVES, &status) )
                {
                /* log error, clean up input queue, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0,
                                                "PushCommand failed");
                        PopRpcInput( (void **)&cmd_args, &status);
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                        clear_RPC_state( );     /* indicate idle on fatals */
                }
                else
                  {
                        argzz.status = E_SUCCESS;
                  }
        }

        if (argzz.status != E_SUCCESS)
        {       /* failure somewhere: free allocated memeory: */
                if (cmd_args) {
                        xdr_free( xdr_RE_recx_file_info, (char *)cmd_args );
                        free( cmd_args );
                }
        }

    set_rpc_obj( re_poll_load_recx_directives, &argzz.RPCobjID );

    return &argzz;
}

/*****************************************************************************
**
** Routine:  re_poll_load_recx_directives_1_svc
**
** Inputs:   RE_null_args
**
** Outputs:  RE_status_result
**
** Purpose:  Function to test for completion of the previously started
**           RE_load_recx_directives operation.
```



```
**
** Intended caller:  Internal Only.
*****************************************************************************
*/

RE_status_result *
re_poll_load_recx_directives_1_svc( IN RE_null_args *arg,
                        IN struct svc_req *req )
{
        static RE_status_result         argzz;
        static RE_status_result         *outarg = NULL;
        int     result, cmd, status;

        if (outarg)
        {       /* free last results */
                xdr_free( xdr_RE_status_result, (char *)outarg );
                free( outarg );
                outarg = NULL;
        }

        /* make sure submit is in progress */
        if ( (argzz.status = check_RPC_state(
                                        FALSE, COMMAND_LOAD_RECX_DIRECTIVES ))
          != E_SUCCESS )
                ;       /* just return failure status */
        /* test for completion of processing: later use real flag */
        else if (PopResult( -1, &result, &cmd, &status) )
        {
                if (status == COMMAND_RECORD_GET_FAILED)
                {
                        argzz.status = EP_RB_RECOVER_RPC_INCOMPLETE;
                }

                else {
                        /* log error, clean up, return error  */
                        EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                              status, 0, "PopResult failed");
                        argzz.status = EP_RB_RECOVER_SERVERFAIL;
                }
        }

        if (argzz.status != E_SUCCESS)
                ;       /* fall thru to error return logic */

        else if (cmd != COMMAND_LOAD_RECX_DIRECTIVES)
        {
                /* log error, clean up, return error  */
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_INVALID_COMMAND, 0,
                        "PopResult mismatch: got %d command, expected %d\n",
                                      cmd, COMMAND_LOAD_RECX_DIRECTIVES);
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (result != COMMAND_RESULT_SUCCESS)
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                      MESSAGE_FAILURE_DOING_ASYNC_RPC, 0,
                                "RPC failure in process manager thread" );
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
        else if (PopRpcOutput( (void **)&outarg, &status) )
        {
                EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR, status,
                                      0, "PopRpcOutput failure");
                argzz.status = EP_RB_RECOVER_SERVERFAIL;
        }
```

```
        else
        {
            /* return popped results struct */
            set_rpc_obj( re_poll_load_recx_directives, &outarg->RPCobjID);
            clear_RPC_state( );
                                        /* indicate process mgr idle */
            return outarg;
        }

        set_rpc_obj( re_poll_load_recx_directives, &argzz.RPCobjID );
        if (argzz.status == EP_RB_RECOVER_SERVERFAIL)
            clear_RPC_state( );
                                /* indicate process mgr idle on fatals */

        return &argzz;
}



/*****************************************************************
 *  RE_get_catalog_info:
 *
 *  This routine returns the level structure with the
 *  level for backup being restored
 *
 * OutPuts:
 *         RE_catalog_info Struct containing The level of the backup,
 *         the number of records, and the catalog type for the backup
 *          being restored
 * Parameters:
 *    RE_time *arg      (I) Time of the backup that is beeing looked at
 *
 * Return Codes: (Stored in argzz.status)
 *              EP_RB_RECOVER_RPC_FAIL - if rpc call failed because the
 *                                       argument was NULL
 *              E_SUCCESS              - if rpc call completed successfully
 *              EP_RB_RECOVER_INVALOP  - if another RPC is running
 *                                       this result is gotten from
 *                                       check_rpc_state
 *****************************************************************/

RE_catalog_info *
re_get_catalog_info_1_svc( IN RE_time *arg,
                           IN struct svc_req *req )
{
       static RE_catalog_info    argzz;
                                /* variable to return to RPC caller*/
       if (
      NULL == arg)  /* we need the input to continue, so if none passed in */
              argzz.status = EP_RB_RECOVER_RPC_FAIL;

       else if ( (argzz.status = check_RPC_state(
                                                FALSE, COMMAND_NONE_ACTIVE ))
         != E_SUCCESS)          /* if RPC not idle, trouble */
              ; /*  /* we weren't idle, reject call */
       else {  /*we are ok to run an RPC :) */
              /* Call the Function to get the catalog info and place
               * its result in the status of the return struct.
               * this call should fill in the required fields
               */
              argzz.status = RSTSL_get_catalog_info( arg->backupTime,
                                                     &argzz.level,
                                                     &argzz.numrec,
                                                     &argzz.catType);
       }
```



```
        set_rpc_obj( re_get_catalog_info, &argzz.RPCobjID );

        return &argzz;
                        /* return our newly retrieved values from the catalog*/
}
```

```
/*
** ========================================================================
** Copyright 1996,1997 EMC Corporation
** ========================================================================
*/
/*
** ========================================================================
** EDMRE_ccr.c
**
** Mission Statement: This is the entry point for the Control Channel Reader
**                    thread. Its main purpose is to read asynchronous
**                    messages from the Dispatch Daemon.
**
**
** Primary Data Acted On:
**
** Compile-Time Options:
**
**                      USE_SUNRPC - Compile source with sunrpc support.  If
**                                   not set, assume DCE support.
**
** Basic idea here: Module for Control Channel Reader thread.
** ========================================================================
*/
/*
** ========================================================================
** The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
** ========================================================================
*/
#if !defined(lint)
static char      RCS_id [] = "@(#)$RCSfile: EDMccr.c,v $ "
                             "$Revision: 1.23 $ "
                             "$Date: 1997/02/06 20:49:15 $" ;
#endif

/* #define _POSIX_SOURCE   unable to compile with this define set */
/* #define _XOPEN_SOURCE   unable to compile with this define set */

#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <esl/inout.h>

#include <csc/cscomm.h>
#include <pthread.h>

// Rogue Wave includes
#include <rw/collect.h>
#include <rw/rwfile.h>
#include <rw/vstream.h>
#include <rw/bintree.h>

#ifdef __cplusplus
extern "C" {
#endif

#include <EDMutils.h>
#include <restore/dispatch_daemon.h>
#include <restore/csc_Dispatch_Protocol_Client.h>
```



```
#include <restore/dispatch_protocol_client.h>
#include <restore/dispatch_protocol_service.h>
#include <restore/dispatch_protocol.h>
#include <dpClient.h>

#ifdef __cplusplus
}
#endif
#include <EDMRE_ccr.h>
#include <EDMSession.h>
#include <logging/logging.h>
#include <EDMReturnMessageApi.h>
#include <EDMRestoreEngLog.h>
#include <EDMDD_ddp.h>

// Global/Extern if_spec to be used by RE_ccw/DD_ccr.
static rpc_if_handle_t if_spec;
static rpc_if_handle_t xtract_if_spec;
rpc_binding_handle_t *restoreService_ccw_handle_p;
DD_client_session_id *p_restoreServiceUid;
static unsigned char *connect_h=NULL;
static void halt_service(int);
static boolean32 print_error = TRUE;

// Internal Function Prototypes
static int edmrst_send_connect_h_to_dd();

void *
RestoreSvc_ccr(void *buff)
{
  int lrc;
  error_status_t status;
  struct sigaction act;
  struct timeval timeout = {5,0};

  //
  // Let begin to listen for requests.
  //
  for(;;)
    {
      lrc = csc_async_server_listen( (esl_timeval*)&timeout, &status);
      if ( 1 != lrc )
       {
           EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_FAILED_LISTEN,
                                0,"csc_async_server_listen() pop");
       }
    } /* End of forever loop */


  //
  //   Unregister our service upon exit request.
  //

  csc_unregister_async_server_interface(&if_spec, &status);
  if ( 1 != lrc )
   {
         EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
                                    DDP_UNREGISTER_SVC, 0,
                               "csc_unregister_async_server_interface(
                                                                ) failure");
   }

  return((void*)0);
}
/*
** ========================================================================
```

```
** Function:    edmrst_send_connect_h_to_dd()
** Description:
**
**
**
** Returns:      0 Sucessful
**              -1 Read Failure
**              <0 Read less than exspected
** ========================================================================
*/
int
edmrst_send_connect_h_to_dd()
{
auto int lrc=0;
auto unsigned char *p_client_h=NULL;
auto error_status_t  status;


//
// Isolate the connection handle from the server 'if_spec'.
// The IP/PORT are part of the created if_spec structure.
//
//lrc = csc_ifspec_get_connect_handle( &if_spec,
//                                     p_client_h,
//                                     &status );

p_client_h = if_spec.connect_handle_p;

//if ( 1 != lrc )
// {
//   EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_IFSPEC_INIT_FAILURE,
//                        0,"csc_ifspec_get_connect_handle() failure");
//   return(-1);
// }

//
// Write the handle to the service so it can contact me
//

lrc = edmrst_WrChannel(STDOUT_FILENO,
                       p_client_h,
                       CONNECT_HANDLE_SIZE);
if ( CONNECT_HANDLE_SIZE != lrc )
  {
    (void) free(p_client_h);
    EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_WRITE_CHANNEL,
                         errno,"edmrst_WrChannel() Failure");

    return(-1);
  }

return(0);
}

/*
** ========================================================================
** Function:    RestoreSvc_Setup()
** Description:
**
**
**
** Returns:      0 Sucessful
**              -1 Read Failure
**              <0 Read less than exspected
** ========================================================================
*/
int
```



```
RestoreSvc_Setup()
{
  int             _dbg=0;
  int             lrc;
  struct hostent *hp;
  struct utsname name;
  error_status_t csc_status;
  int             status;


    while(_dbg) {
        ;
    }

  //
  // Setp the server ifspec
  //
  lrc = csc_async_ifspec_init (&if_spec,
                               CSC_IFSPEC_PRIVATE_TYPE,
                               DP_PROGNUM,
                               DP_VERSNUM,
                               (
                          dispatch_func_p_t)&edm_dispatch_protocol_client_1_table,
                               &csc_status);

  /* make sure that the initialization of the structure was ok */
  if ( TRUE != lrc )
  {
      EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_IFSPEC_INIT_FAILURE,
                           csc_status,"csc_async_ifspec_init() Failure");
      return(-1);
  }


  //
  // Read the svc handle from the stdin descriptor.
  // The port in this handle is the one we need to
  // connect to to contact the dispatch RDR thread.
  //
  lrc = edmrst_get_client_handle( STDIN_FILENO,
                                  &connect_h );

  if ( 0 != lrc )
  {
      EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_GET_CLIENT_HANDLE,
                           0,"edmrst_get_client_rpc_handle() failure");
      return(-1);
  }

  //
  // Read the Unique Session ID from the stdin descriptor.
  //
  lrc = edmrst_read_uid_from_channel( STDIN_FILENO,
                                      (void**)&p_restoreServiceUid,
                                      sizeof(DD_client_session_id) );
  if ( 0 != lrc )
  {
      EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_GET_UID_FAILURE,
                           errno,"edmrst_read_from_target_channel() failure");
      return(-1);
  }

//  lrc = csc_ifspec_init( &xtract_if_spec,
//                         EDM_DISPATCH_PROTOCOL_SERVICE,
//                         EDMDPS_FUNCTIONS,
//                         NULL );
//  if ( 1 != lrc )
```

```
//  {
//    EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_IFSPEC_INIT_FAILURE,
//                         0,"csc_ifspec_init() Failure");
//    return(-1);
//  }

  //
  // Extract the port that we will meet on. The port being
  // used is to connect the dispatch CCR with the restore
  // service CCW.
  //
  lrc = csc_private_ifspec_init(connect_h,
                                EDM_DISPATCH_PROTOCOL_SERVICE,
                                EDMDPS_FUNCTIONS,
                                &xtract_if_spec,
                                &csc_status);

  if ( 1 != lrc )
  {
      EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_IFSPEC_INIT_FAILURE,
                           0,"csc_private_ifspec_init() failure");
      return(-1);
  }

  //
  //  We need the system name and ip for the if_spec.
  //
  uname( &name );
  hp = gethostbyname( name.nodename );
  if ( NULL == hp )
  {
      EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_GETHOSTNAME_FAILURE,
                           errno,"gethostbyname() failure");
      return(-1);
  }
  ( void ) memcpy( (char*) &if_spec.ip_addr, hp->h_addr, hp->h_length );


  //
  // Register service with csc layer.
  //
  lrc = csc_register_async_server_interface(&if_spec,
                                            -1,
                                            edm_dispatch_protocol_client_1_table,
                                            edm_dispatch_protocol_client_1_nproc,
                                            &csc_status);

  if ( 1 != lrc )
    {
      EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,DDP_REGISTER_SVC_FAILURE,
                           0,"csc_register_async_server_interface() failure");
      return(-1);
    }
  if ( IsDebugOn() )
    EDMRestoreEng_logent(__FILE__,__LINE__,LOG_INFO,0,
                         0,"PORT_INFO if_spec(RECCR) port#: %d",
                         if_spec.portnum);

  restoreService_ccw_handle_p = (rpc_binding_handle_t *)
                                 calloc(1, sizeof(rpc_binding_handle_t));
  //
  // Create the client connection handle to be used by the
  // restore service ccw thread to send messages to the
  // dispatch daemon.
  //
  lrc = csc_connect_to_async_rpc_service((char*)NULL,
                                         xtract_if_spec,
                                         restoreService_ccw_handle_p,
                                         &csc_status);
  if ( 1 != lrc )
    {
      EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,
                           DDP_PRIVATE_SVC_CONNECT_FAILURE,
                           0,"csc_connect_to_rpc_service() failure");
      return(-1);
    }
  if ( IsDebugOn() )
     EDMRestoreEng_logent(__FILE__,__LINE__,LOG_INFO,0,
                          0,"PORT_INFO xtract_if_spec(RECCW) port#: %d",
                          xtract_if_spec.portnum);
  //
  //  Send the Restore Service port/ip to the Dispatch Daemon.
  //  This information will be used by the Dispatch Daemon CCW
  //  thread when sending messages to this restore service.
  //
  lrc = edmrst_send_connect_h_to_dd();
  if ( 0 != lrc )
    {
      EDMRestoreEng_logent(__FILE__,__LINE__,LOG_ERR,
                           DDP_SEND_CONNECT_HANDLE_FAILURE,
                           0,"edmrst_send_connect_h_to_dd() failure");
      return(-1);
    }

  //
  // Queue the initial connect indicate message.
  //
   lrc = PushResponseMessage(dp_connect_indicate,
                             *p_restoreServiceUid,
                             restoreService_ccw_handle_p,
                             &status);

return(0);
}
```

```
/*****************************************************************************
**
** File Name:   RSLinitfin.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**              This module contains the Restore Service Library functions to
**              initialize and terminate the restore operation.
**
** Table of Contents:
** ------------------
**              RSTSL_Initialize
**              RSTSL_Finish
**
**      Internal Functions:
**
**
** Compile-Time Options:
**              This section must list any compile time definitions
**              which will affect this header.
**
*****************************************************************************/


/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */
#include <sys/param.h>
#include <dirent.h>
#include <dlfcn.h>


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>


/*
 * Local headers
 */
```



```
#include <RSLinterns.h>
#include <RSLstartup.h>


/*
 * #defines, structures, typedefs local to this source file
 */

static eerrno_ty init_plugins( restore_context *rcp );
static int validate_plugin( struct pluginData *piDataPtr );


/*
 * External declarations
 *
 * This is the global "restore context" that will be used
 * throughout the rest of the restore operations.
 */
struct restore_context *rcp = NULL;


/*
 * Definitions of the names of the plugin functions in the piFuncArray
 * of the pluginData structure. These must be in the same order and position
 * as the PIFuncIndex values defined in RSLplugin.h.
 */

char *piFuncNames[PIFuncIndexLast+1] = {
        "RSTPI_Identify",
        "RSTPI_Initialize",
        "RSTPI_GetTopLevelObjects",
        "RSTPI_SetTopLevelObject",
        "RSTPI_GetNextLevelObjects",
        "RSTPI_ClearRestoreContext",
        "RSTPI_Submit",
        "RSTPI_GetTopLevelTemplates",
        "RSTPI_DoesAlternateExist",
        "RSTPI_MarkObject",
        "RSTPI_UnmarkObject",
        "RSTPI_IsObjectMarked",
        "RSTPI_IsObjectMarkable",
        "RSTPI_GetAllBackupTimes",
        "RSTPI_GetCurrentBackupTime",
        "RSTPI_SetBackupForTime",
        "RSTPI_SetPrevBackup",
        "RSTPI_SetNextBackup",
        "RSTPI_SetFirstBackup",
        "RSTPI_SetMostRecentBackup",
        "RSTPI_IsTherePrevBackup",
        "RSTPI_IsThereNextBackup",
        "RSTPI_IsTherePrevBackupForTime",
        "RSTPI_IsThereNextBackupForTime",
        "RSTPI_Finish",
        "RSTPI_StartRestore",
        "RSTPI_FindRestorableObjects",
        "RSTPI_GetFindResults",
        "RSTPI_GetNecessaryMedia"
};
```

```
/*****************************************************************
 * RSTSL_Initialize:
 *
 *  This function takes care of all the initialization for a restore
 *  session.  This must be called prior to any of the other functions
 *  in the Restore API.
 *
 * Parameters:
 *
 *  userName (I) - The name of the user.
 *
 *****************************************************************/

eerrno_ty
RSTSL_Initialize( const char *userName )
{
    eerrno_ty status = E_SUCCESS;

    /*
     * If we have not yet allocated space for a restore_context
     * structure, do so now.  If we have already done so, just clear it
     * now.
     */

    if (NULL == rcp)
    {
        rcp = (struct restore_context *)malloc(sizeof(struct restore_context));
        if (NULL == rcp)
         {
             rec_api_log_csm(SUB_CSM_NOMEM, NULL);
             return(EP_RB_RECOVER_NOMEM);
         }
    }
    memset(rcp, 0, sizeof(struct restore_context));

    rcp->rc_human_uidname = esl_strdup( userName );


    if (!rcp->rc_human_uidname) {
      rec_api_log_csm(SUB_CSM_NOMEM, NULL);
      return(EP_RB_RECOVER_NOMEM);
    }

    /*
     * Set the appropriate field in the recovery context to indicate
     * that this recover session is based on the Recover API.
     * This flag is in place for historical reasons but is used by
     * other parts of the Recover API library.
     */

    rcp->gui_mode = 1;

    /*
     * Initialize the logging mechanism.
     */

    if (status = rbrlog_begin(rcp, progname))
    {
        return(status);
    }

    /*
     * Initialize the few "recover context" variables that we can at
     * this early stage.
     */
```



```
    setup_proc(rcp);

    /*
     * The following call will:
     *   -Initialize the saveset database.
     *   -Infer any information we can at this point.
     */

    if (status = startup(rcp))
    {
        return(status);
    }

    /* Do plugins setup: Find and initialize all valid restore plugin libs: */

    status = init_plugins( rcp );

    return( status );
}       /* End of RSTSL_Initialize() */


/*****************************************************************
 * RSTSL_Finish
 *
 *    Function Description:
 *
 *  This function terminates a restoral session,
                                                but not while a restore is in
 *  progress. It will be rejected if a restore is currently being executed.
 *  This routine will clean up any local memory used in the session.
 *
 * Parameters:
 *
 *  none
 *
 */

eerrno_ty
RSTSL_Finish( void )
{
    int mc_n;


    eerrno_ty err = E_SUCCESS;

    if (NULL == rcp)
    {
        return( E_SUCCESS );
    }
    RemoveSubmitFiles();
    /*
     * Call rbr_cleanup() which kills the aux proc(s), unlocks the work
     * item, then calls rbrlog_end() to enter the last logs and to close
     * the log file.
     */

    rbr_cleanup(rcp);

    /*
     * Deallocate the memory of restore_context and the related
     * structures.
     */

    if (NULL != rcp->rc_mcp)      /* Free the multicat structures */
    {
        mcat_destroy(rcp->rc_mcp);
```

```
    }

    /*
     * Free the mark bit map space
     */

    for (mc_n = 0; mc_n < rcp->rc_marks_plane_alloc; mc_n++)
    {
        if (NULL != rcp->rc_marks[mc_n])
        {
            free(rcp->rc_marks[mc_n]);
        }
        rcp->rc_marks[mc_n] = NULL;
    }

    if (NULL != rcp->rc_nmarks_by_plane)
    {
        free(rcp->rc_nmarks_by_plane);
    }

    /*
     * Free the configuration structures
     */

#if 0
    if (NULL != rcp->rc_cfgname)
    {
        free(rcp->rc_cfgname);
    }
#endif

    if (NULL != rcp->rc_config)
    {
        rbc_freeconfig(rcp->rc_config);
    }

    /*
     * Free the DS_NONE structures array
     * Note that even though rc_dsnones is the head of linked list
     * of dsnone_info structures, the list is allocated via malloc
     * as an array initially (ref. alloc_plane_arrays()), therefore
     * we can do a free here.
     */

    if (NULL != rcp->rc_dsnones)
    {
        free(rcp->rc_dsnones);
    }

    /*
     * Free the volume list structures.
     */

    if (NULL != rcp->ebvllist)
    {
        (void)ebvl_volidlist_destructor(rcp->ebvllist, EBVL_DESTROY_ALL);
    }

    /*
     * Free the plugin related data
     */

    rcp->rc_backup_app = 0;
    while (rcp->currentPIptr = rcp->piList)
    {
        rcp->rc_backup_app++;
```



```
        rcp->appData = rcp->currentPIptr->appData;
        /* allow plugin to clean up and close .so: */
        if ( E_SUCCESS != (err =
            rcp-> currentPIptr-> piFuncArray[ PIFuncIndexFinish ] (rcp) ) )
        {
        /* log error, continue */
            rbe_user_error( err,
                    "RSTPI_Finish failed for restore plug-in library
                                                                    %s\n",
                ((struct pluginIDdata *)(
                                    rcp-> currentPIptr-> idData))-> name );
        }
        dlclose( rcp-> currentPIptr-> libHdl );
        rcp->piList = rcp->piList->next;
        free (rcp->currentPIptr);
    }

    /*
     * Free the various simple string buffers
     */

    if (NULL != rcp->rc_top_level_object_name)
    {
        free(rcp->rc_top_level_object_name);
    }

    if (NULL != rcp->rc_template_name)
    {
        free(rcp->rc_template_name);
    }

    if (NULL != rcp->rc_workitem_name)
    {
        free(rcp->rc_workitem_name);
    }

    if (NULL != rcp->rc_human_uidname)
    {
        free(rcp->rc_human_uidname);
    }

    if (NULL != rcp->rc_effective_uidname)
    {
        /* don't free, its internal: free(rcp->rc_effective_uidname); */
    }

    if (NULL != rcp->rc_client_rbuname)
    {
        free(rcp->rc_client_rbuname);
    }

    if (NULL != rcp->rc_client_hostname)
    {
        free(rcp->rc_client_hostname);
    }

    if (NULL != rcp->rc_client_scriptname)
    {
        /* don't free, its internal: free(rcp->rc_client_scriptname); */
    }

    if (NULL != rcp->rc_client_dirtop)
    {
        free(rcp->rc_client_dirtop);
    }
```

```
    if (NULL != rcp->rc_cmd_context)
    {
        /* don't free -- its internal/temp data: free(rcp->rc_cmd_context);
                                                                    */
    }

    if (NULL != rcp->rc_source_client_hostname)
    {
        free(rcp->rc_source_client_hostname);
    }

    if (NULL != rcp->rc_cpiogen_executable)
    {
        /* don't free, its internal: free(rcp->rc_cpiogen_executable); */
    }

    if (NULL != rcp->rc_plugin_wi_types)
    {
        free(rcp->rc_plugin_wi_types);
    }

    if (NULL != rcp->rc_pwd)
    {
        free(rcp->rc_pwd);
    }

    /*
     * Finally, deallocate the restore_context itself
     */

    free(rcp);
    rcp = NULL;



    return( err );
}       /* RSTSL_Finish */



/*****************************************************************************
 * init_plugins
 *
 *    Function Description:
 *
 * This function locates, opens, validates and initializes all restore
 * plug-in (shared) libraries.  They must be located in
 * /usr/epoch/EB/cure_plugin (eb_cure_plugin_dir).  All .so files in that
 * directory are opened and validates for version# and presence of all
 * mandatory functions.  The RSTPI_Identify function is called for each
 * library to determine which optional features are supported, and that
 * the corresponding functions are present.  Finally, the RSTPI_Initialize
 * function is called for each valid library.
 *
 * Parameters:
 *
 * Inputs:
 *       rcp     (I)     - Pointer to restore context
 *
 * Outputs:
 *       none
 *
 * Returns:
 *       E_SUCCESS or EP_RB_RECOVER_xxx
 *
 * Logic/pseudo code:
```



```
 *
 *       open plugin dir
 *       while read_next_entry succeeds
 *               verify .so file (else continue)
 *               open shared library file (else continue)
 *               on errors below:
 *                       close shared library file
 *                       continue
 *               fetch all mandatory function addresses
 *               call Identify function
 *               validate version number
 *               fetch all indicated optional function addrs
 *               call Initialize function
 *               add workitem types to composite exclusion list
 *               add to valid plugin list
 *       close plugin dir
 */

static eerrno_ty init_plugins( restore_context *rcp )
{
        DIR             *dirp;
        struct dirent   *direntp;
        eerrno_ty       status = E_SUCCESS;
        struct pluginData *piDataPtr = NULL;
        struct pluginData *piListPtr = NULL;
        int             val_result;
        struct pluginIDdata *idDataPtr;
        char            *tmp_types;
        int             shlib_dirlen;
        char            shlib_path [MAXPATHLEN];

        /* open plugin directory or bust */
        if ( NULL == (dirp = opendir( eb_cure_plugin_dir )) )
        {
                rec_api_log_csm( SUB_CSM_PLUGIN_ERR, NULL );
#if 1
                return E_SUCCESS;       /* allow continuation w/o plugins */
#else
                return EP_RB_RECOVER_NO_PLUGINS;        /* later do this */
#endif
        }

        strcpy( shlib_path, eb_cure_plugin_dir );
        strcat( shlib_path, "/" );
        shlib_dirlen = strlen (shlib_path);

        /* loop thru entries in directory*/
        while (NULL != (direntp = readdir( dirp )))
        {
                if (NULL == piDataPtr)
                {       /* allocate next plugin data structure */
                        if (NULL == (piDataPtr
                                        = calloc( 1, sizeof(
                                                        struct pluginData) )))
                        {
                                status = EP_RB_RECOVER_NOMEM;
                                break;          /* fall thru to cleanup */
                        }
                }

                if (NULL == strstr( direntp->d_name, ".so" ) )
                        continue;       /* skip this guy */

                strcpy( &shlib_path[shlib_dirlen], direntp->d_name );
                if (NULL == (piDataPtr->libHdl
                                = dlopen( shlib_path, RTLD_NOW )))
```

```
            {
                rbe_user_error( 0,
                    "Error opening restore plug-in library %s: %s\n",
                                direntp->d_name, dlerror() );
                continue;       /* skip this one */
            }

        /* Fetch addresses of all mandatory functions and */
        /* Do Identify processing: call it, save options, validate */

            if (0 != (val_result = validate_plugin( piDataPtr ) ))
            {
                if (val_result == -1 || val_result == -4)
                {
                    rbe_user_error( 0,
                "Functions missing from restore plug-in library %s: %s\n",
                                direntp->d_name, dlerror() );
                }
                else if (val_result < 0)
                {
                    rbe_user_error( 0,
                      "Validation failed for restore plug-in library
                                                              %s\n",
                                direntp->d_name );
                }
                else
                {
                    rbe_user_error( val_result,
                "RSTPI_Identify failed for restore plug-in library %s\n",
                                direntp->d_name );
                }

                dlclose( piDataPtr->libHdl );  /* close .so on errors */
                piDataPtr->libHdl = NULL;
                continue;   /* on any error, skip this lib */
            }

        /* let DC plug-in do its initialization */
            rcp->appData = NULL;    /* enter plugin with clean appdata */
            status = piDataPtr->piFuncArray[PIFuncIndexInitialize]( rcp );
            if (E_SUCCESS != status)
            {
                rbe_user_error( status,
                "RSTPI_Initialize failed for restore plug-in library %s\n",
                                direntp->d_name );
                dlclose( piDataPtr->libHdl );
                                        /* close .so on errors */
                piDataPtr->libHdl = NULL;
                status = E_SUCCESS;     /* this was't fatal */
                continue;       /* on any error, skip this lib */
            }

        /* save plugin's appData */
            piDataPtr->appData = rcp->appData;
            rcp->appData = NULL;

        /* add piDataPtr to valid plugin list */
            if (NULL == piListPtr)
                rcp->piList= piDataPtr;
                                                /* first in list */
            else
                piListPtr->next = piDataPtr;    /* link from prev */
            piListPtr = piDataPtr;              /* new end of list */
            piDataPtr = NULL;

        /* add workitem types to composite exclusion list */
```



```
            idDataPtr = (struct pluginIDdata *)piListPtr->idData;
            if (idDataPtr->num_types > 0)
            {
                tmp_types = calloc( 1, 1 + idDataPtr->num_types
                                        + rcp->rc_num_plugin_wi_types );
                if (NULL == tmp_types) {
                    status = EP_RB_RECOVER_NOMEM;
                    break;
                }
                if (NULL != rcp->rc_plugin_wi_types)
                { /* move old list to new buffer and free old list */
                    memcpy( tmp_types,
                            rcp->rc_plugin_wi_types,
                            rcp->rc_num_plugin_wi_types );
                    free( rcp->rc_plugin_wi_types );
                }
              memcpy( tmp_types + rcp->rc_num_plugin_wi_types,
                      idDataPtr->wi_types,
                      idDataPtr->num_types );
              rcp->rc_num_plugin_wi_types += idDataPtr->num_types;
              tmp_types[rcp->rc_num_plugin_wi_types] = 0;
              rcp->rc_plugin_wi_types = tmp_types;
            }
        }

        (void)closedir( dirp );

        /* free up leftovers: */
        if (NULL != piDataPtr)
                free (piDataPtr);

        if (E_SUCCESS != status)
        { /* Free contents of plugin list: */
                while (NULL != (piDataPtr = piListPtr))
                { /* allow plugin to clean up and close .so: */
                        rcp->appData = piDataPtr->appData;
                        piDataPtr->piFuncArray[PIFuncIndexFinish]( rcp );
                        dlclose( piDataPtr->libHdl );
                        piListPtr = piDataPtr->next;
                        free (piDataPtr);
                }
        }

        return status;
}
        /* init_plugins */

/*****************************************************************
 * validate_plugin
 *
 *    Function Description:
 *
 *  This function retrieves the addresses of the mandatory plugin functions
 *  and stores them in the function pointer array.  If any function is missing
 *  it returns -1.
                        It then calls the identify function and verifies wthe plugin
 *  version, and finds its optional functions. Specific error values are
 *  returned on version mismatch and missing optional functions.
 *
 * Parameters:
 *
 * Inputs:
 *       piDataPtr (I)   - Pointer to plugin data structure with libHdl set
 *
 * Outputs:
 *       piFuncArray in piDataPtr is loaded with pointers to plugin functions
```

```
 *
 * Returns:
 *       0 on success
 *       -1 on any missing required functions
 *       -2 if version validation fails OR identify returns junk
 *       -3 if workitem type validation fails
 *       -4 on any missing optional functions indicated by options flags
 *       +n (
 *          EB_RB_RECOVER_xxx) for error codes returned from Identify function
 *
 ***********************************************************************/

static int validate_plugin( struct pluginData *piDataPtr )
{
        int             index;
        eerrno_ty       status;
        struct pluginIDdata *idDataPtr;

        for( index = 0; index <= PIFuncIndexLastBasic; index++ )
        {
                if (NULL == (piDataPtr->piFuncArray[index]
                        = (piFuncPtr) dlsym( piDataPtr->libHdl,
                                             piFuncNames[index] ))
                )
                        return -1;
        }

        /* call identify and validate: */
        status = piDataPtr->piFuncArray[PIFuncIndexIdentify](
                                                        &piDataPtr->idData
                                                                );
        if (status != E_SUCCESS)
                return status;
        if (NULL == (idDataPtr = (struct pluginIDdata *)piDataPtr->idData) )
                return -2;

        if (idDataPtr->version != RSTPI_VERSION)
        { /* only version 1 supported so far */
                piDataPtr->idData = NULL;
                return -2;
        }

        if (idDataPtr->num_types && !idDataPtr->wi_types)
        {       /* count cant be positive with null pointer */
                piDataPtr->idData = NULL;
                return -3;
        }

        /* if startRestore option set, get its addr or bust */
        if ( ( (RSTPI_OPTION_SPECIAL_START
                == (idDataPtr->options & RSTPI_OPTION_MASK_START))
            && (NULL == (piDataPtr->piFuncArray[PIFuncIndexStartRestore]
                        = (piFuncPtr) dlsym( piDataPtr->libHdl,
                                        piFuncNames[PIFuncIndexStartRestore]
                                                                ))))
          )
        /* OR if special find option set, get its addr or bust */
          || ( (RSTPI_OPTION_SPECIAL_FIND
                == (idDataPtr->options & RSTPI_OPTION_MASK_FIND))
            && ( (NULL == (piDataPtr->piFuncArray[PIFuncIndexFind]
                        = (piFuncPtr) dlsym( piDataPtr->libHdl,
                                             piFuncNames[PIFuncIndexFind]
                                                                )))
              || (NULL == (piDataPtr->piFuncArray[PIFuncIndexFindResults]
                        = (piFuncPtr) dlsym( piDataPtr->libHdl,
```



```
                                        piFuncNames[PIFuncIndexFindResults])
                        ) ) )           )
          /* OR if special getMedia option set, get its addr or bust */
            || ( (RSTPI_OPTION_SPECIAL_GET_MEDIA
                == (idDataPtr->options & RSTPI_OPTION_MASK_GET_MEDIA))
              && (NULL == (piDataPtr->piFuncArray[PIFuncIndexGetMedia]
                        = (piFuncPtr) dlsym( piDataPtr->libHdl,
                                             piFuncNames[PIFuncIndexGetMedia]
                                                                )))
        )
        {
                piDataPtr->idData = NULL;
                return -4;
        }

        return 0;
}
/*      validate_plugin */
```